Spatial data structures

© 1998-2015 Josef Pelikán
CGG MFF UK Praha

pepca@cgg.mff.cuni.cz
http://cgg.mff.cuni.cz/~pepca/

Application areas

- **geographic information systems** (GIS)
  - – area, line and point entities
  - – huge databases (**10^4** to **10^9** objects)
- **image analysis, recognition**
- **computer graphics, games**
  - – 2D & 3D algorithm speedup (ray casting, collisions)
- **industry, CAD**
  - – VLSI design, component positioning, collisions

Elementary tasks I

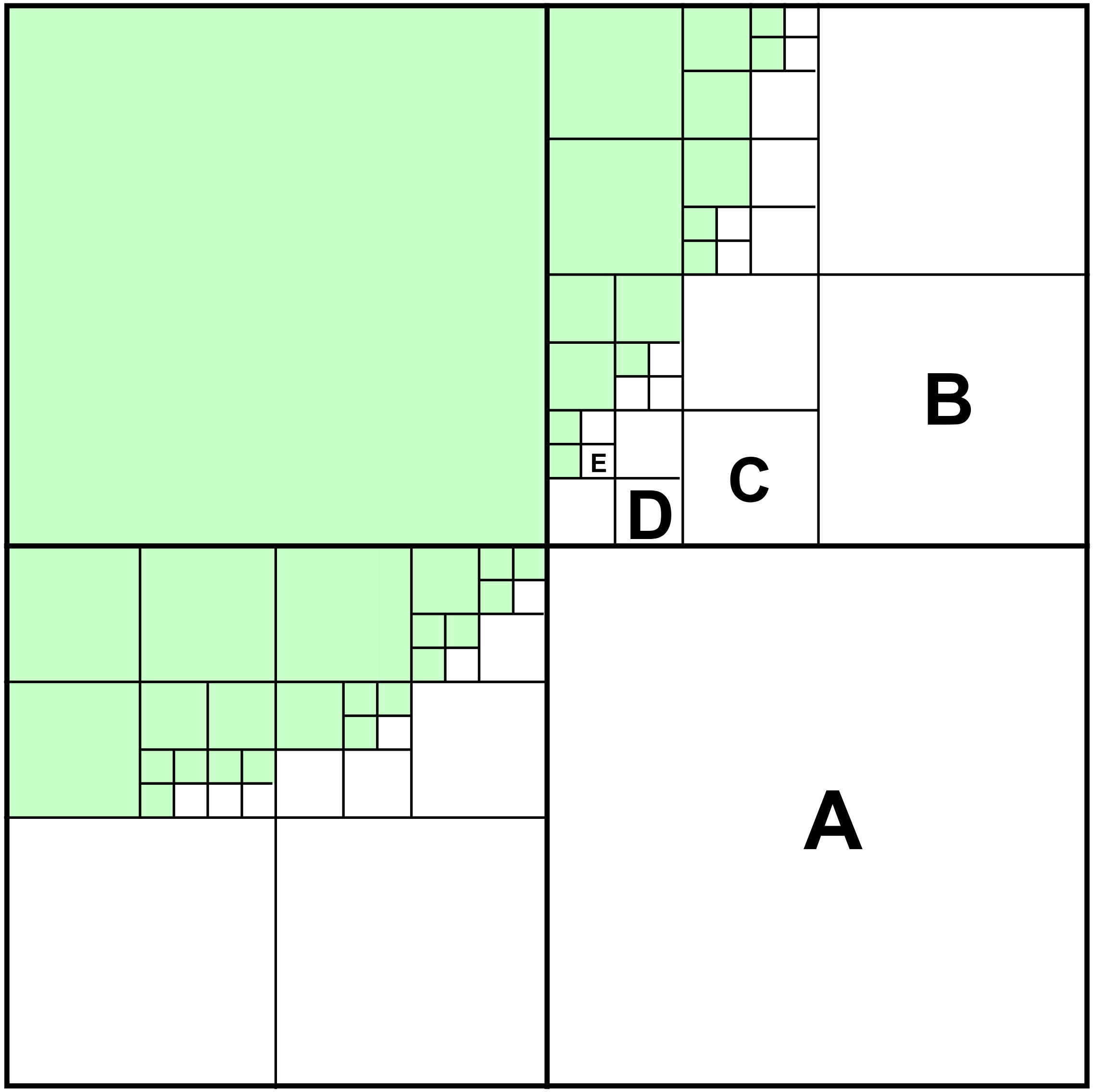
- **point localization** in 2-3D net
  – looking for an area object which contains the point
- **nearest N points** from the given center
  – global variation: looking for the closest point pair
- **curve intersections** (polylines)
  – collisions in a set of curves (polylines)
- **point objects closest to the given curve** (route)

Elementary tasks II

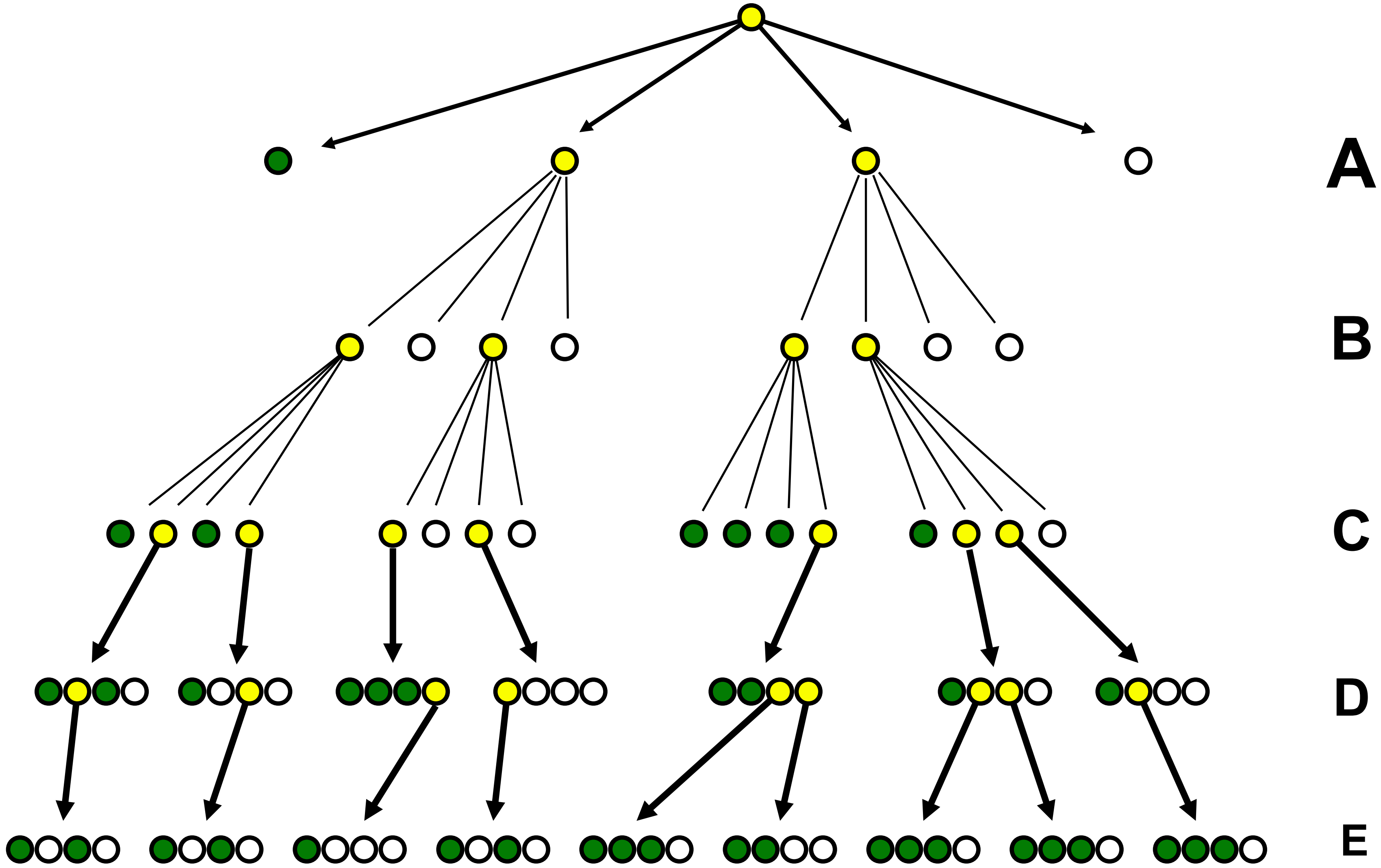
- **interval queries** in 2-3D space (databases)
- **set operations** on map entities
  - – areas, line objects, points
- **collision tests (interferences)**, minimal distances among planar objects (VLSI)
- first **ray-scene intersection** (3D scene)
- object processing in some **geometric order**
  - increasing distance from a given center

„Region quadtree"

- representation of <u>areal regions</u> in 2D
- splitting exactly <u>in the middle</u>
- information is stored <u>in leaf nodes only</u>

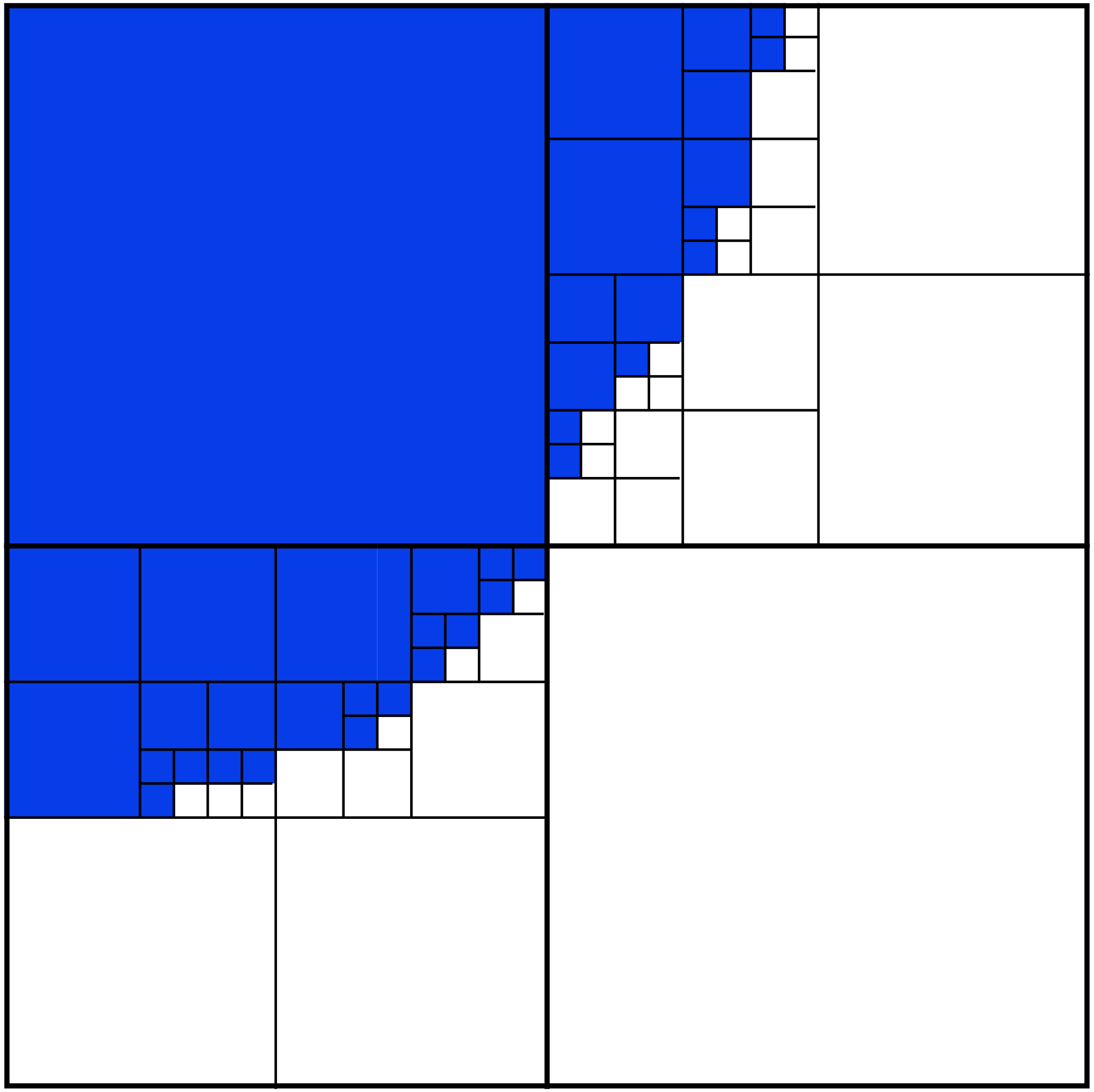


„Region quadtree"



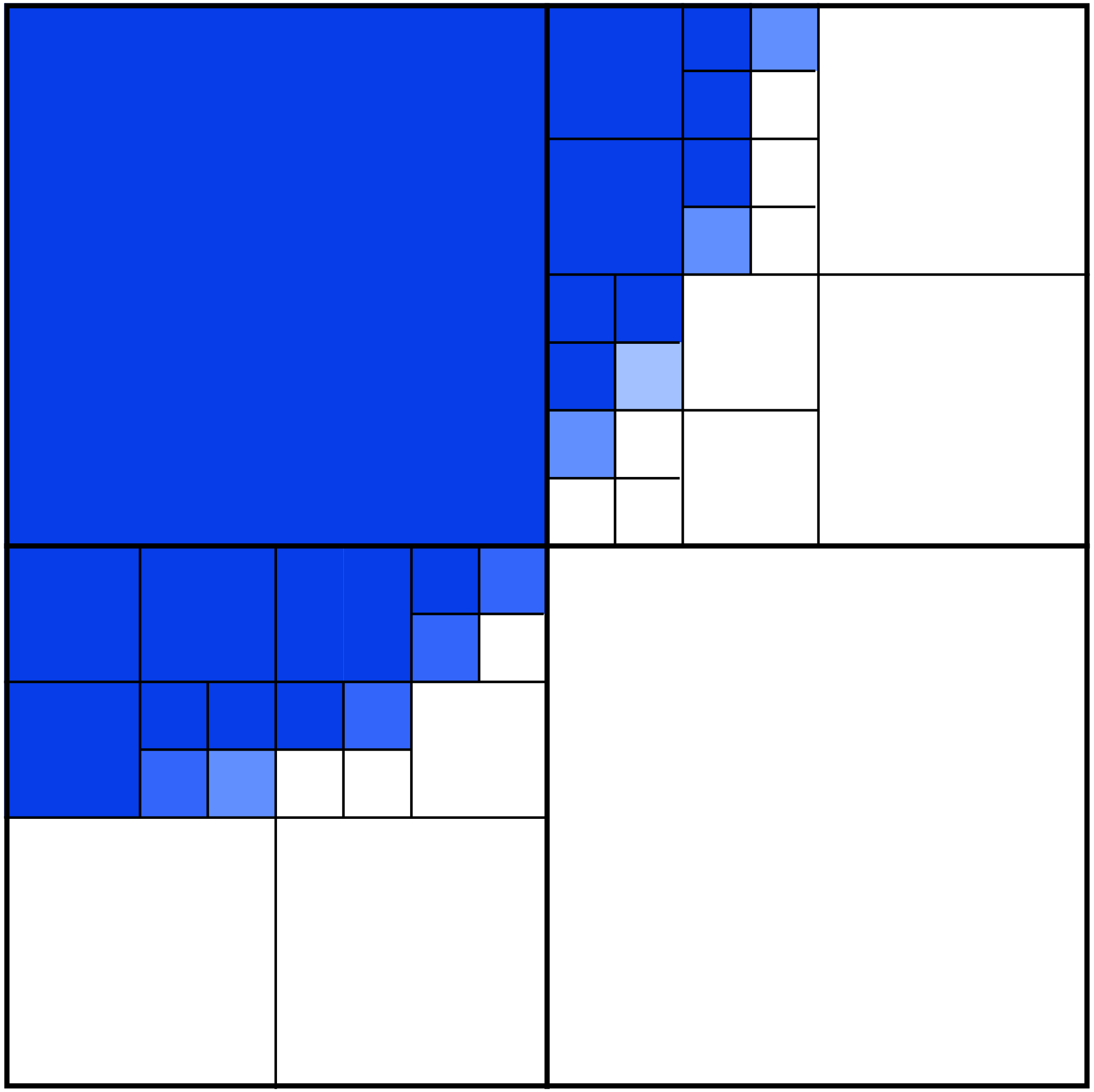
Pyramid

- representation of <u>areal regions</u> in 2D
- additional <u>summary info in inner nodes</u>
- pyramid nodes up to level: E



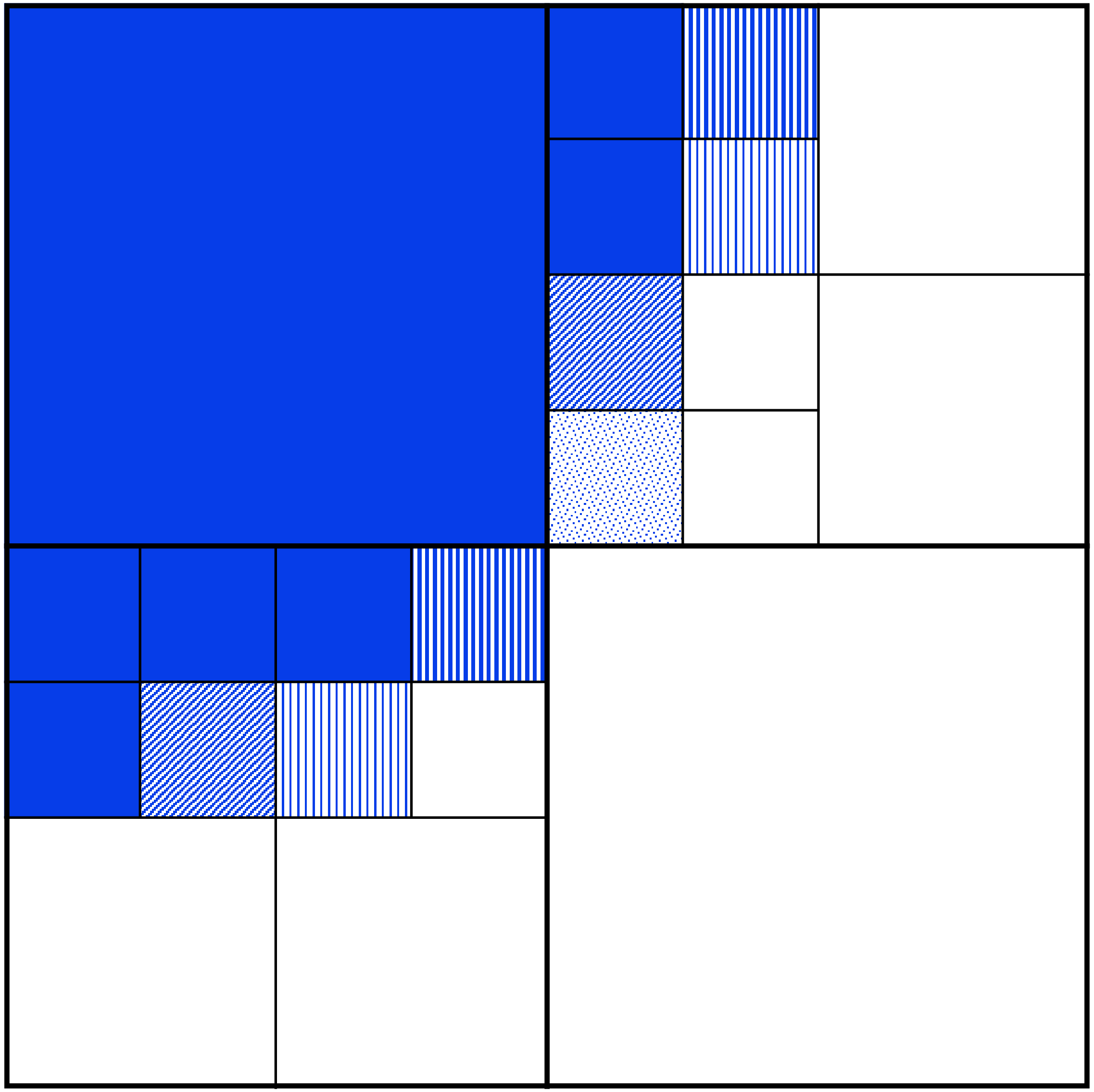
Pyramid

- **pyramid nodes up to level: D**



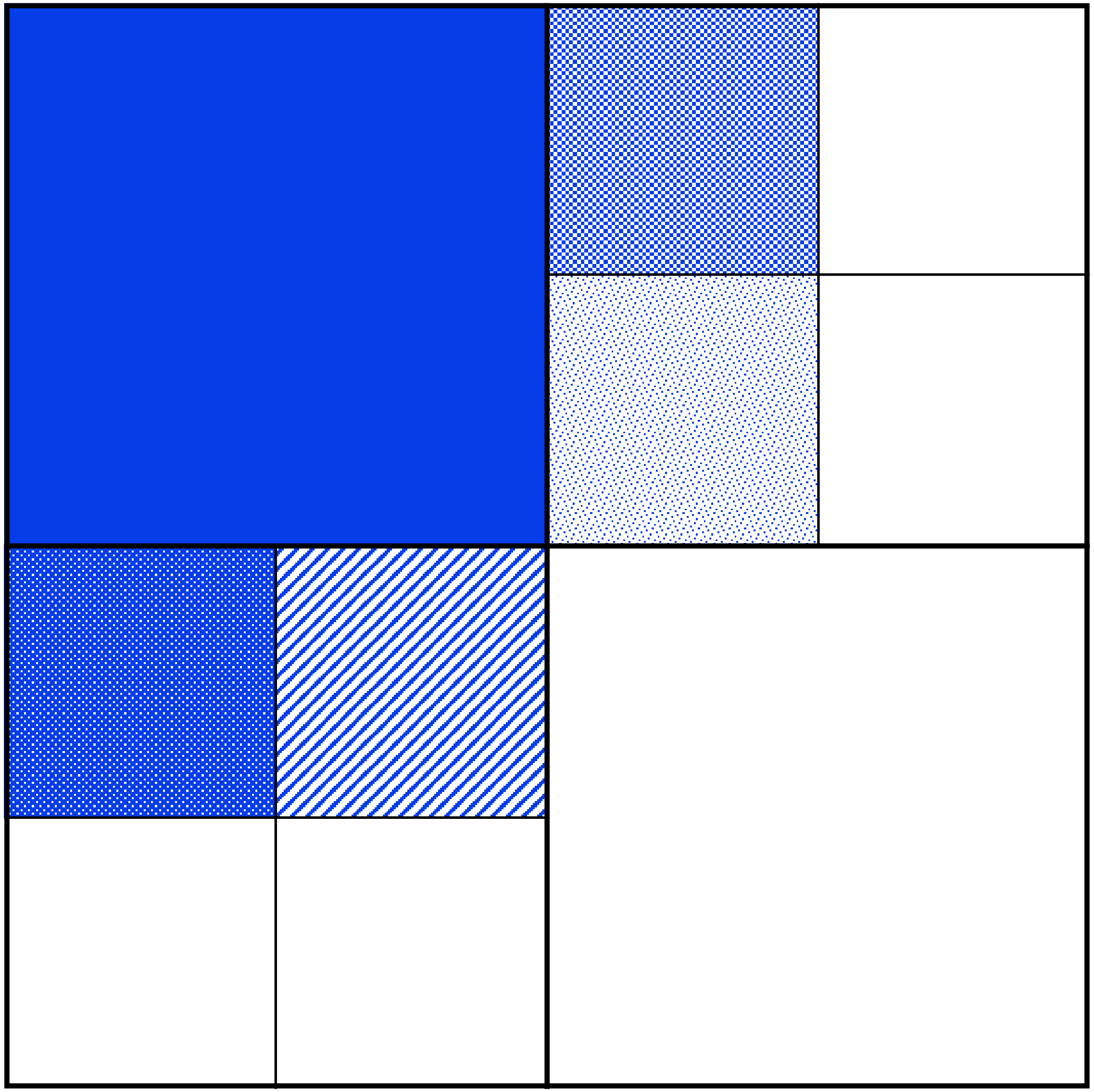
Pyramid

- **pyramid nodes up to level: C**



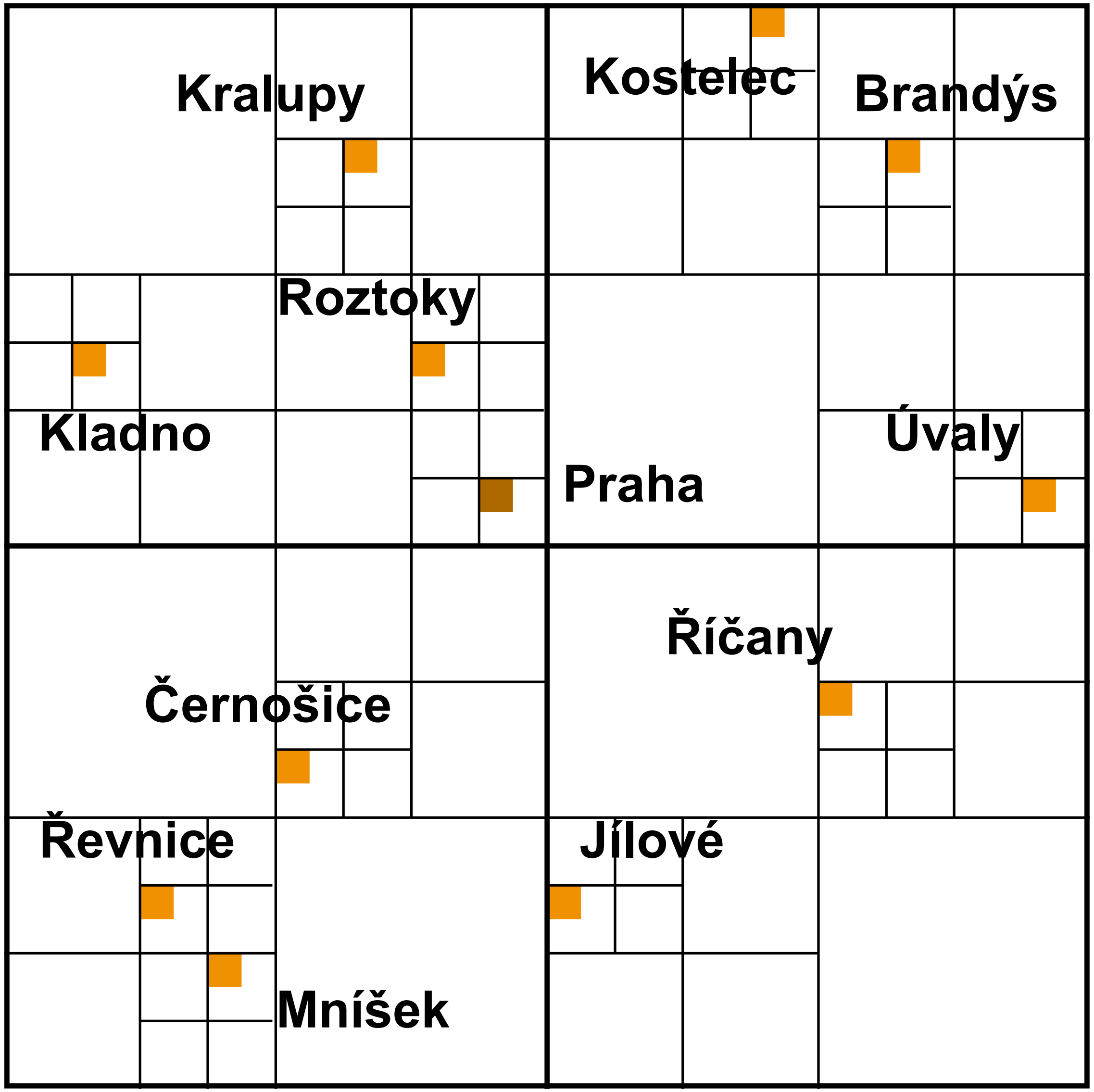
Pyramid

- **pyramid nodes up to level: B**



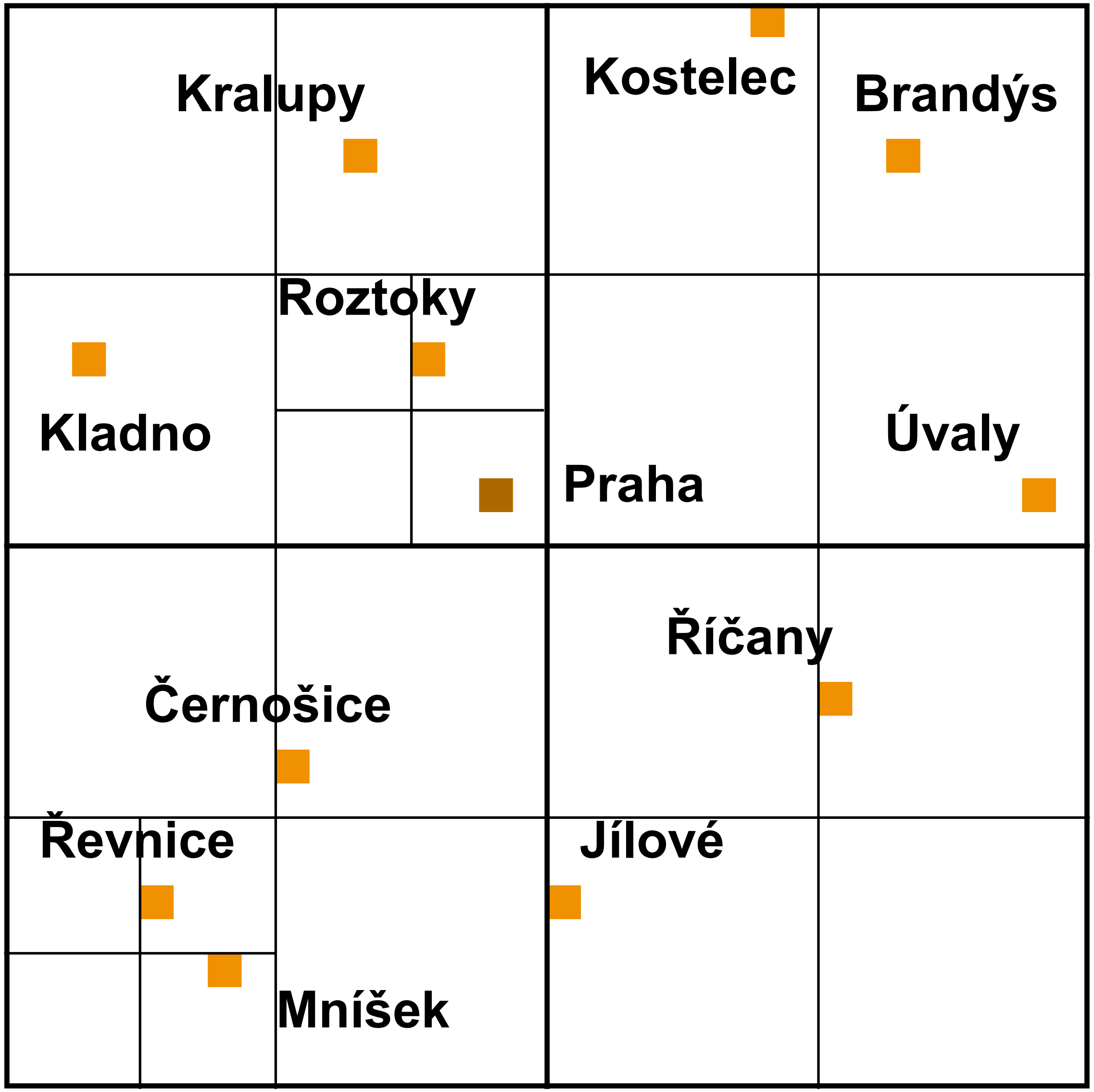
„MX quadtree” (MatriX)

- representation of **point objects**
- splitting exactly **in the middle**
- object information stored **in the leafs** (**of equal depths**)

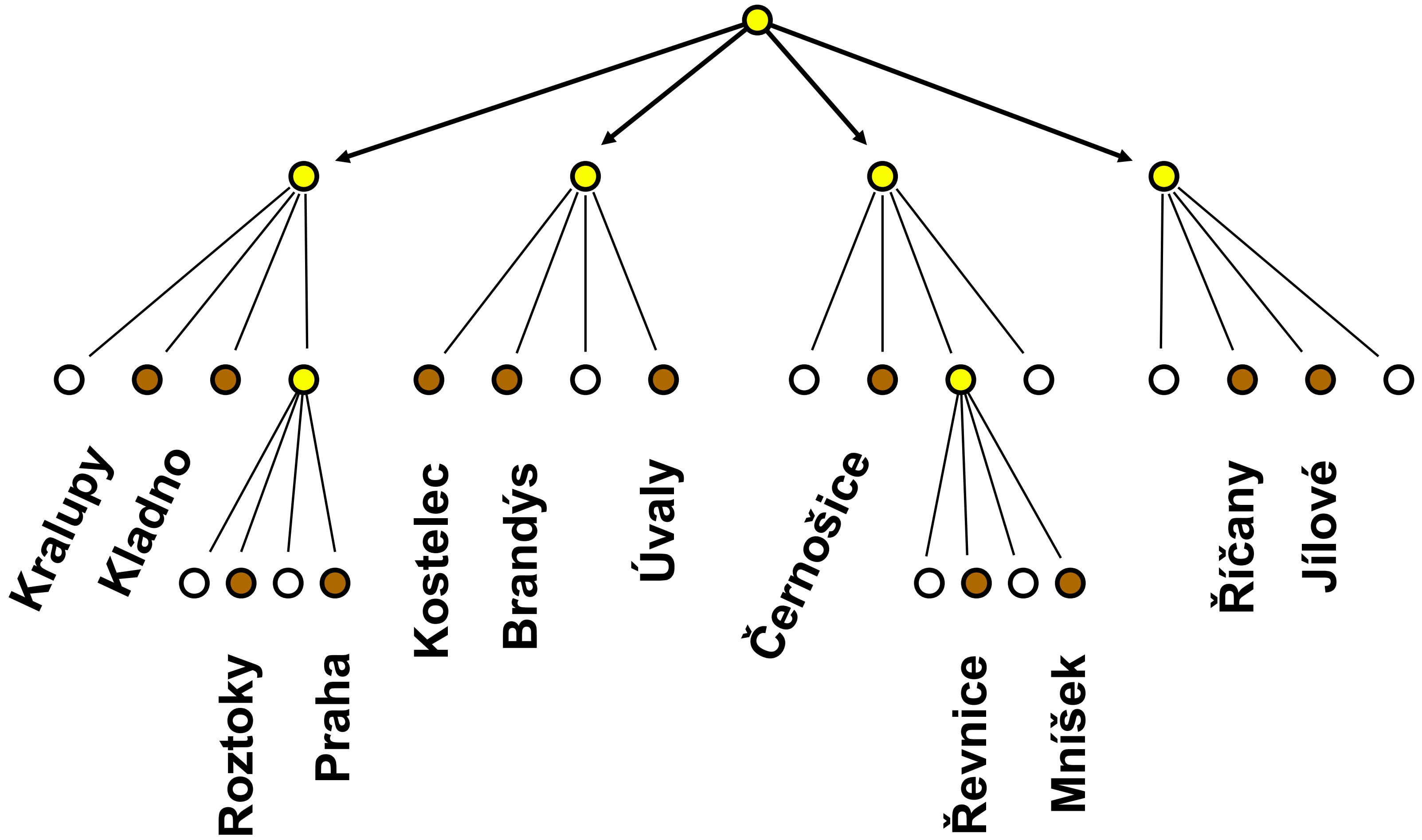


„PR quadtree” (Point-Region)

- representation <u>of point objects</u>
- splitting exactly <u>in the middle</u>
- object information stored <u>in leaf nodes</u> (<u>all levels, max. one object per node</u>)



„PR quadtree” (Orenstein)



Bucket methods

- node can store information about **more than one object**

  $0 < \text{object\#} \leq \text{Max}$

  – **Max** is constant value tuned for storage efficiency (record size..)

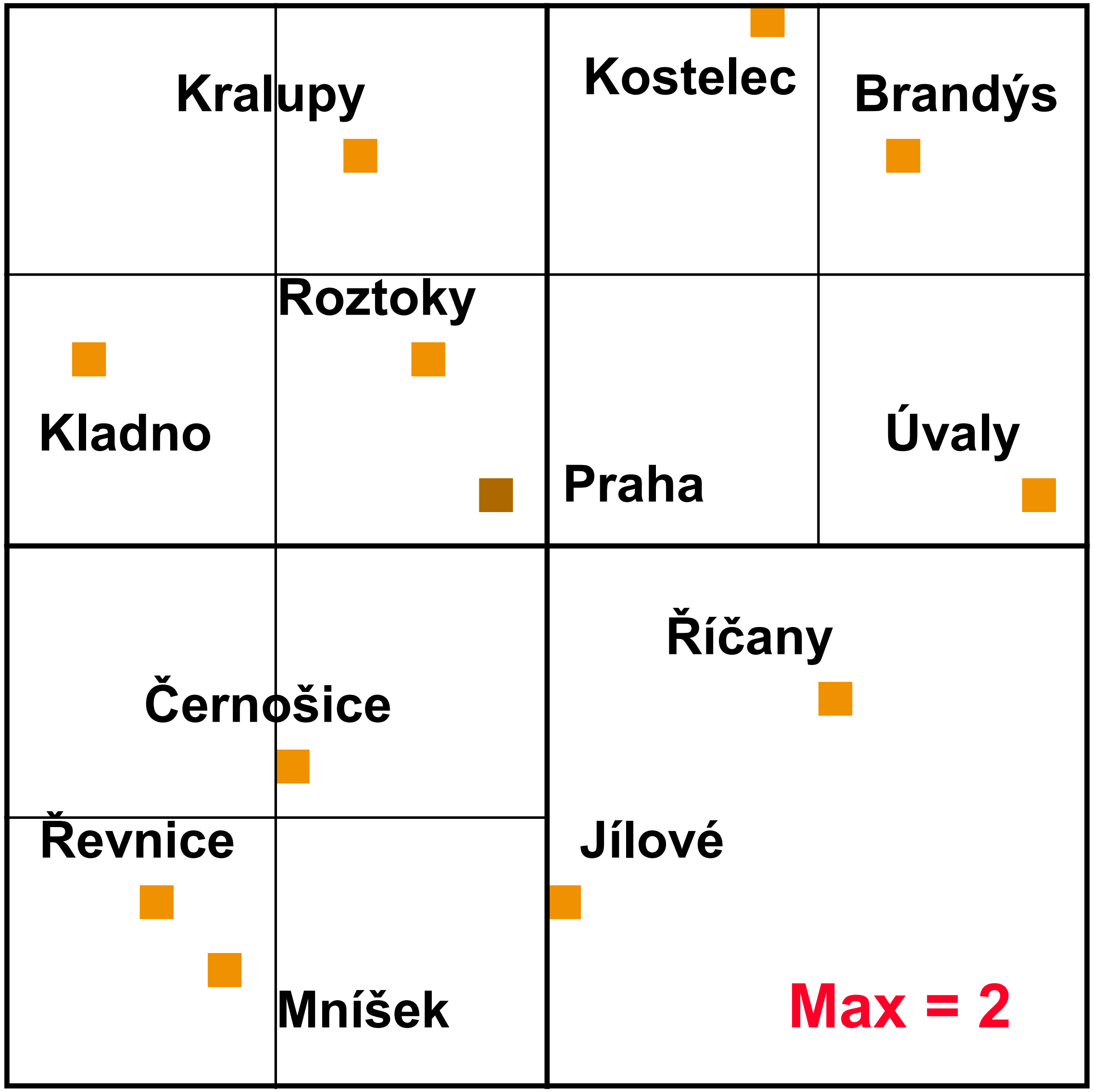
- **memory & time** efficiency

  – less overhead in pointer management

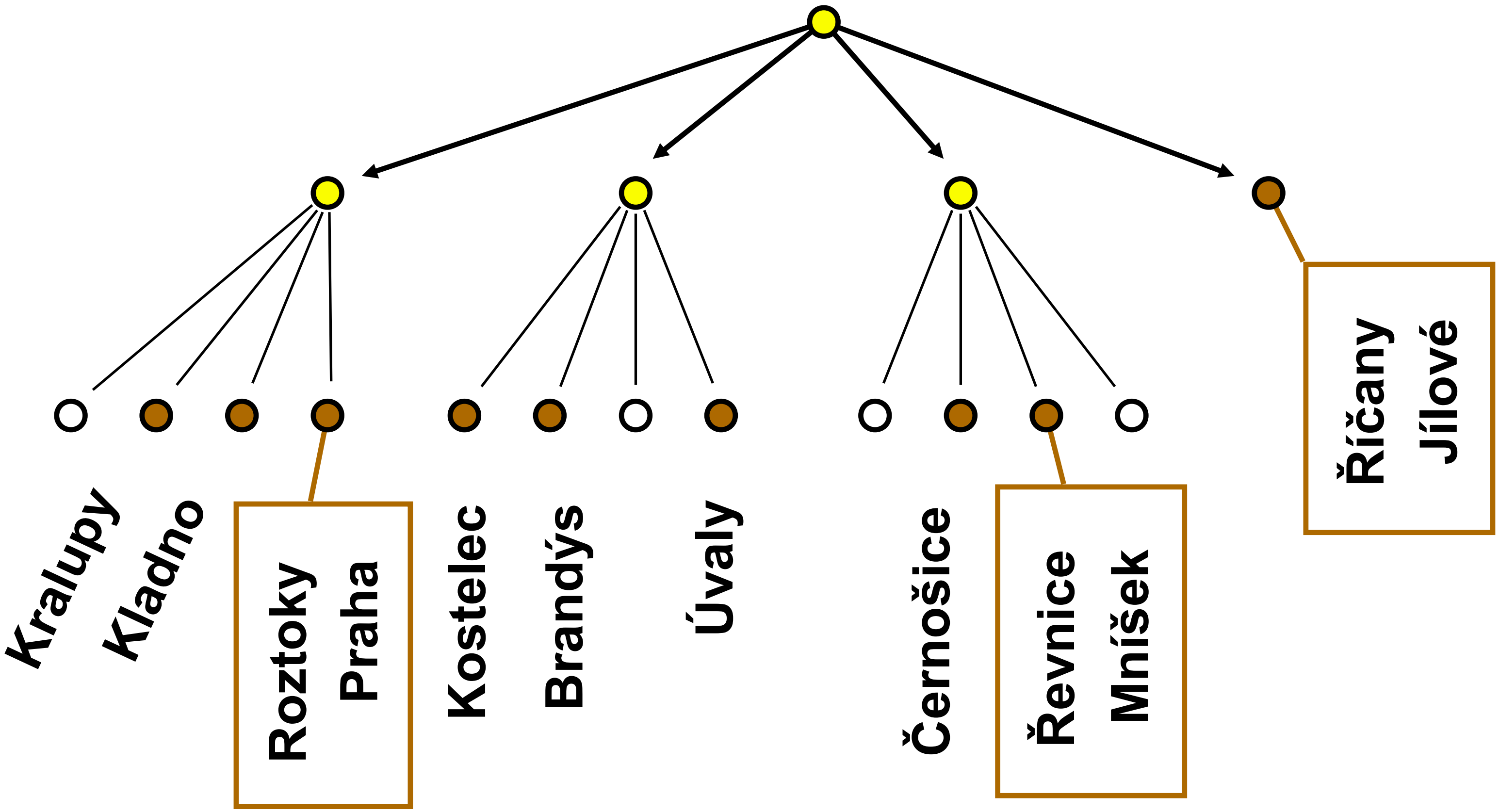
- **AB-tree** analogy

„Bucket PR quadtree”

- representation <u>of point objects</u>
- splitting exactly <u>in the middle</u>
- object information stored <u>in leaf nodes</u> (<u>all levels, ≤ Max object per node</u>)



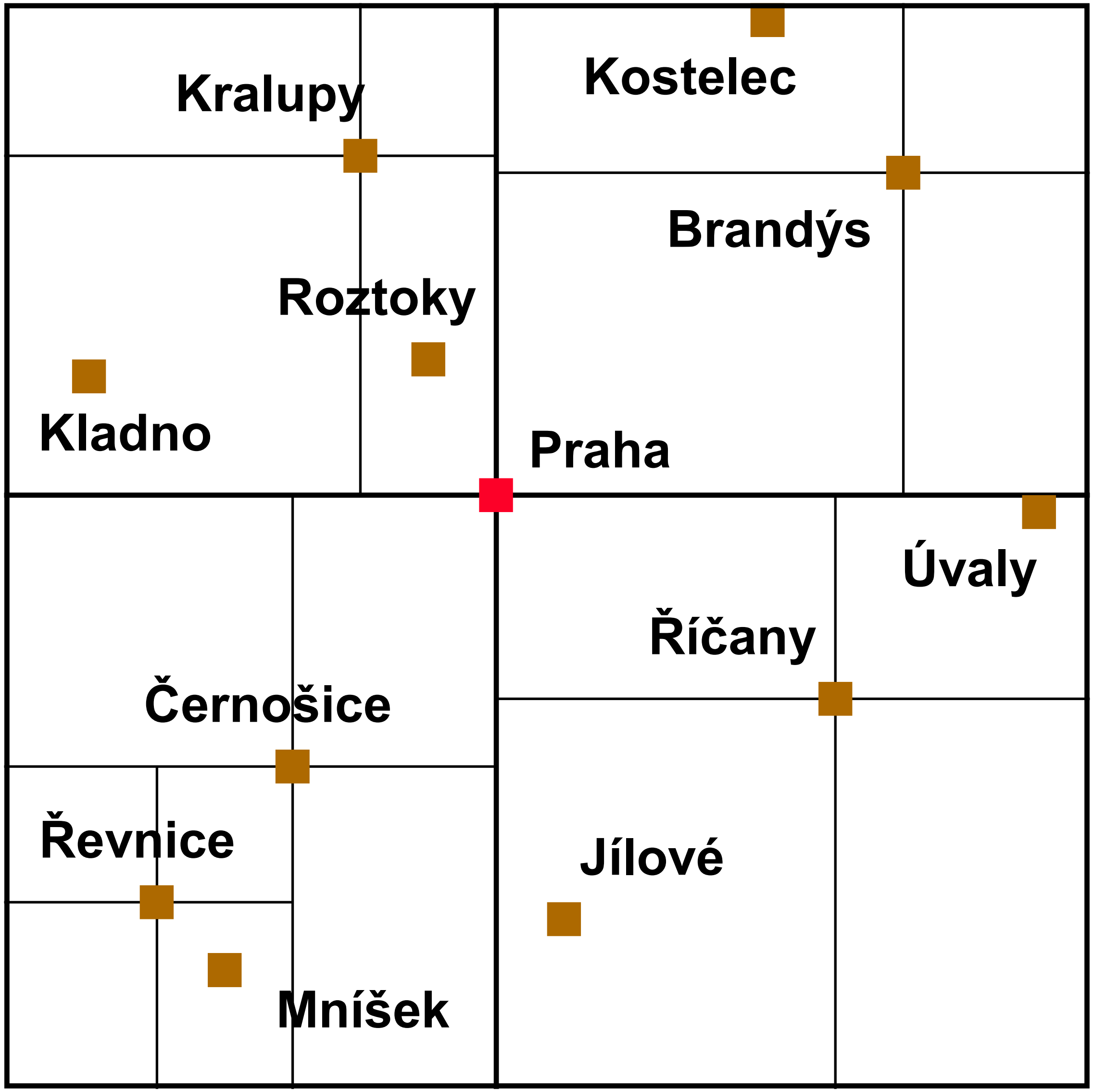
„Bucket PR quadtree”



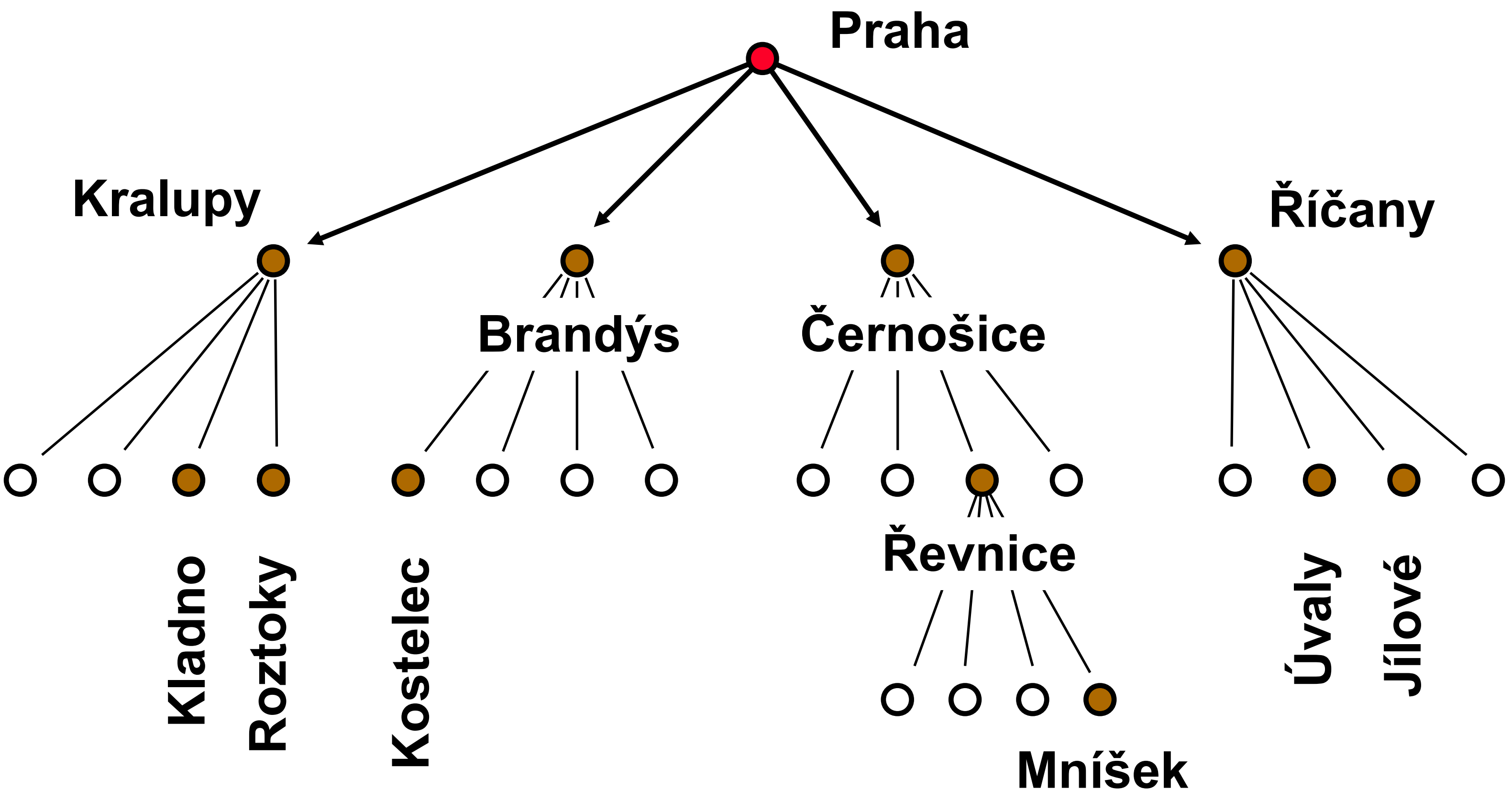
Max = 2

„Point quadtree” (Finkel/Bentley)

- representation of point objects
- adaptive splitting (exactly at object position)
- object information stored in all nodes (even inner ones)

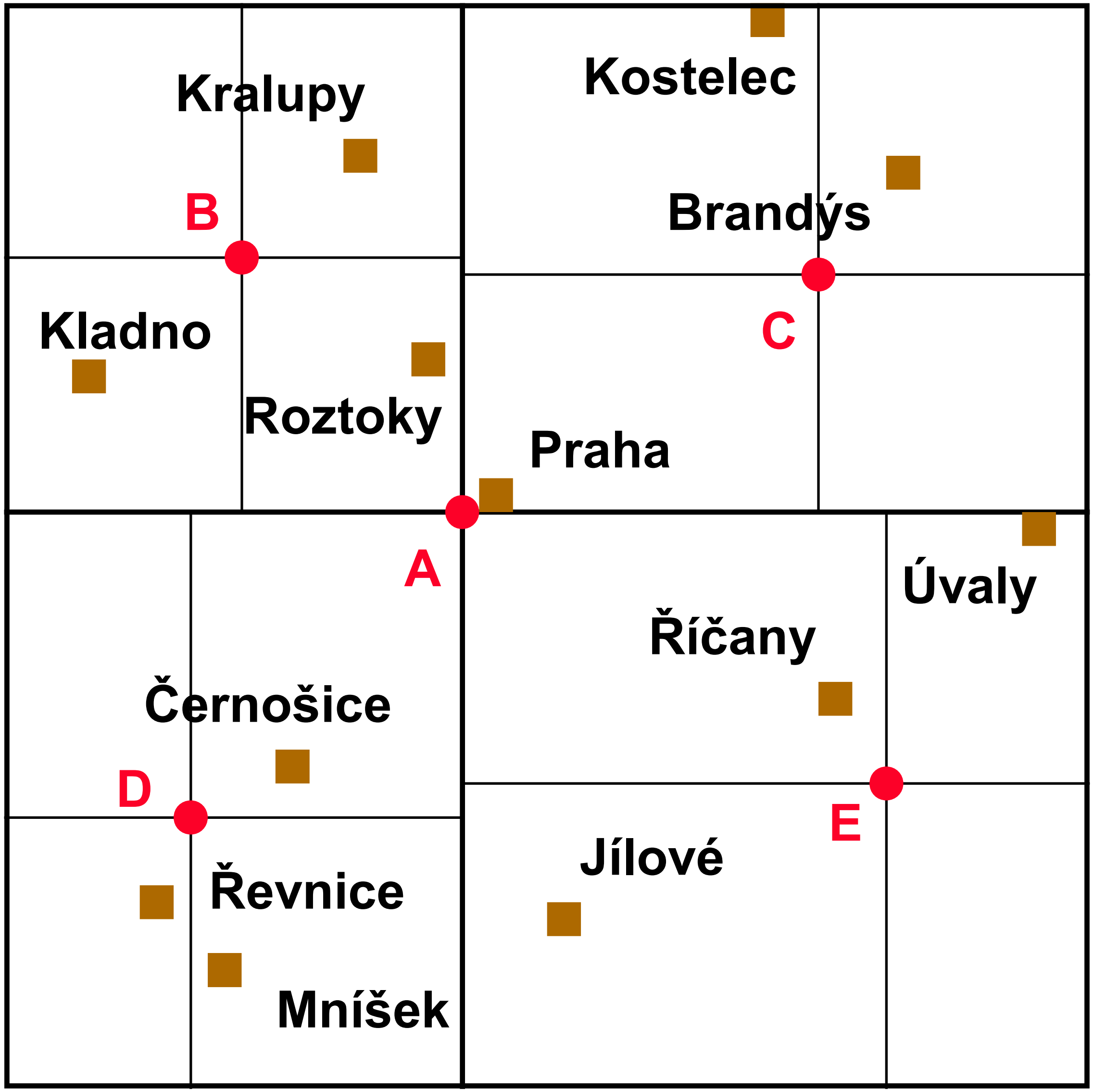


„Point quadtree”

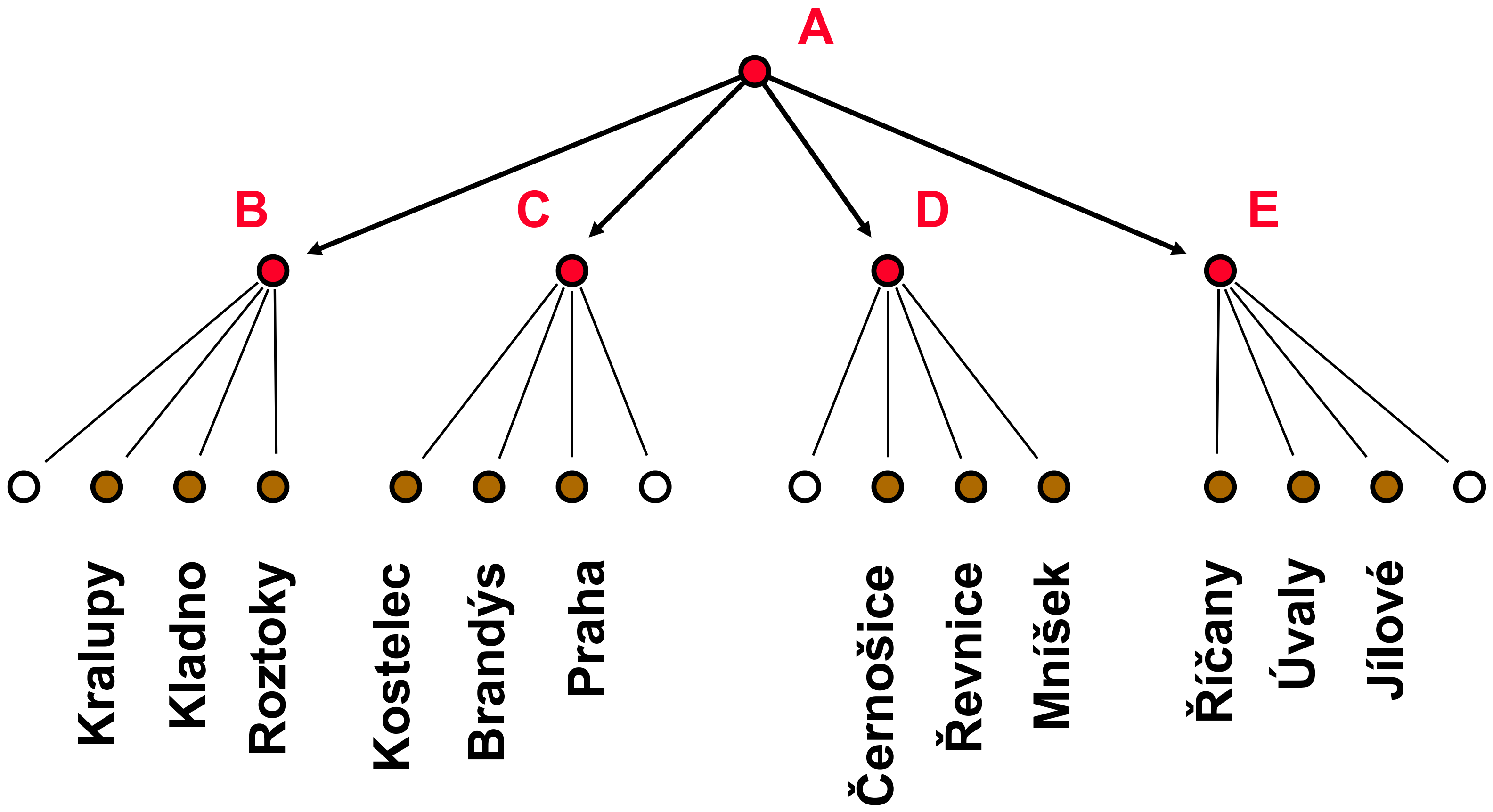


„Pseudo quadtree” (Overmars)

- representation of point objects
- adaptive splitting (loosely by object position)
- object information stored in leafs only

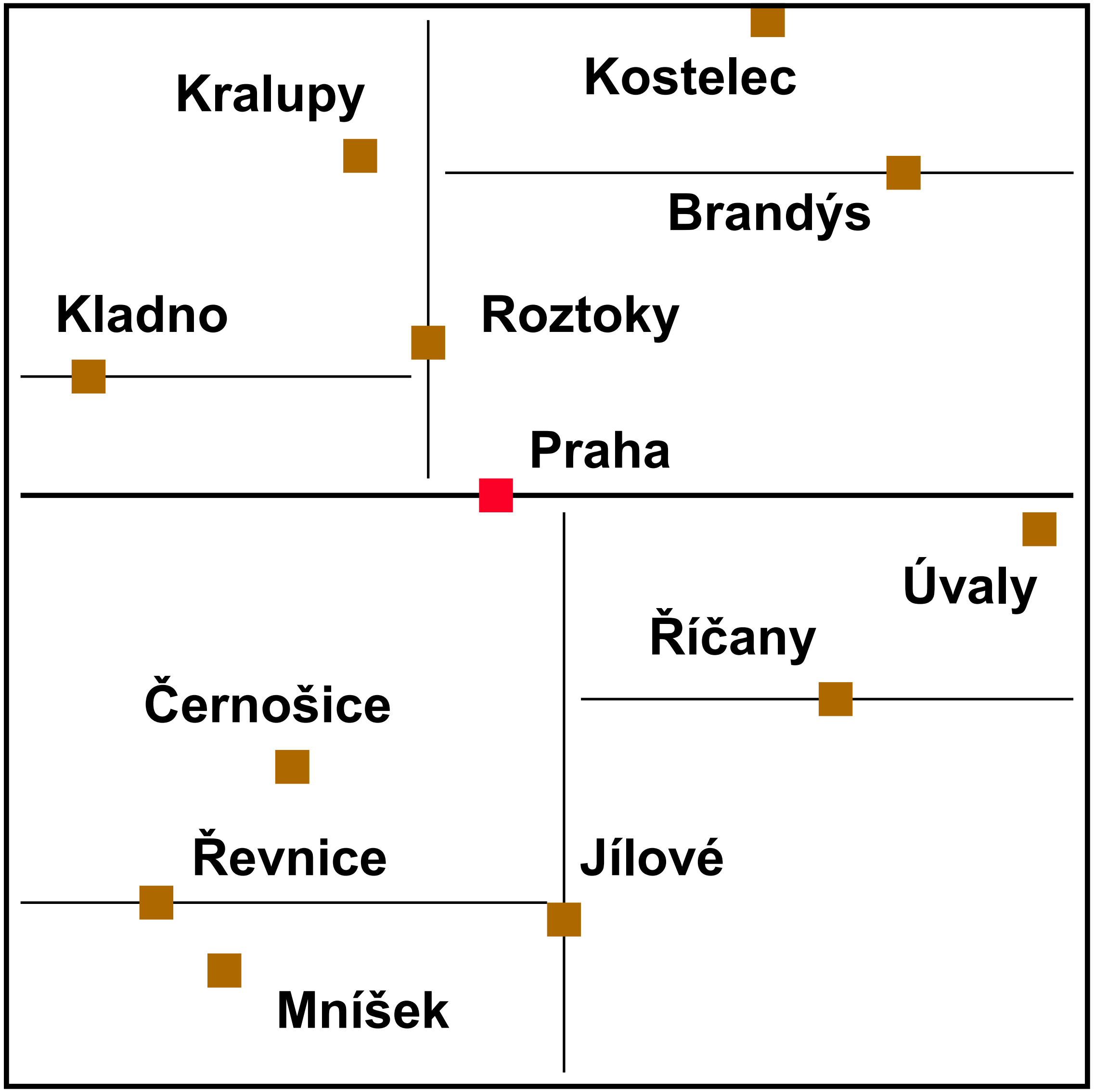


„Pseudo quadtree”

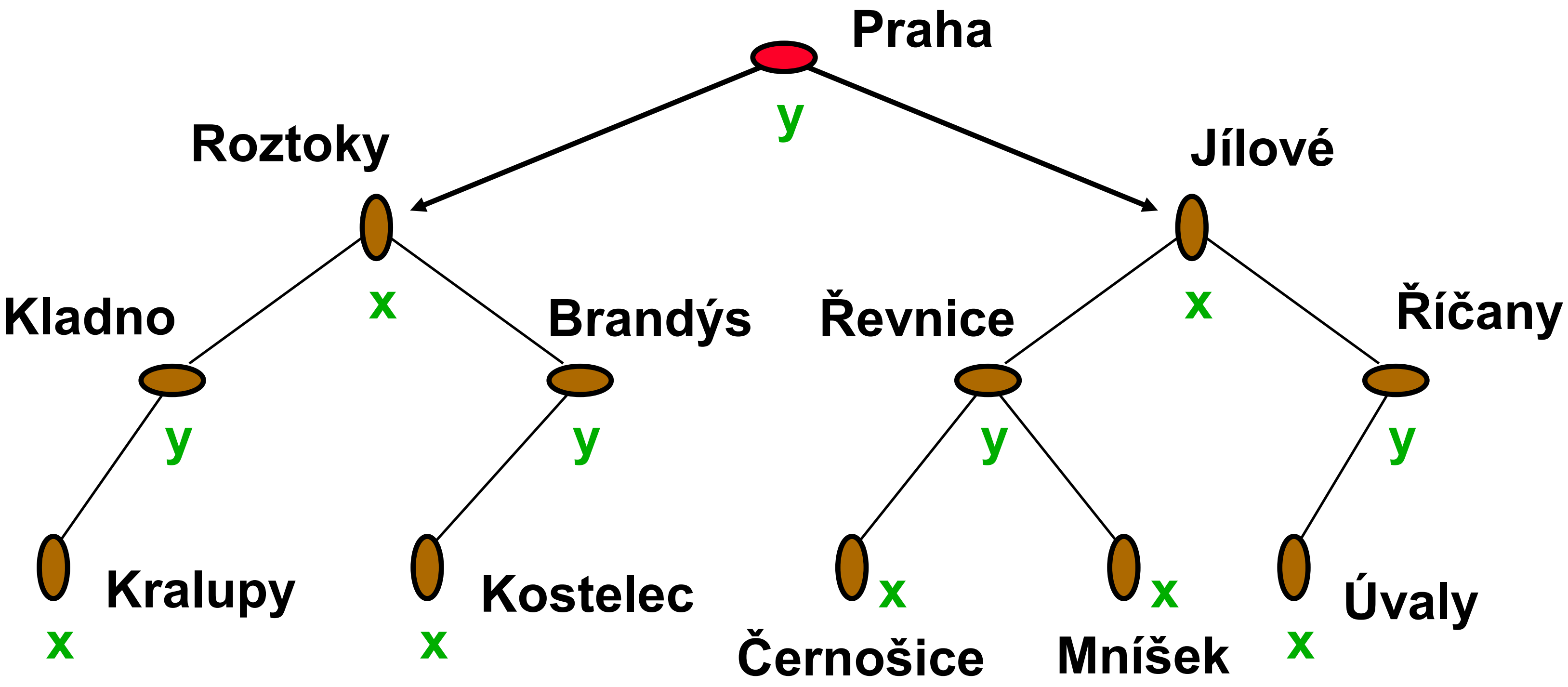


„K-D tree” (Bentley)

- representation **of point objects**
- adaptive splitting - **one coordinate at a time (binary tree)**. regular alternation of the coordinates
- object information stored in **all nodes**

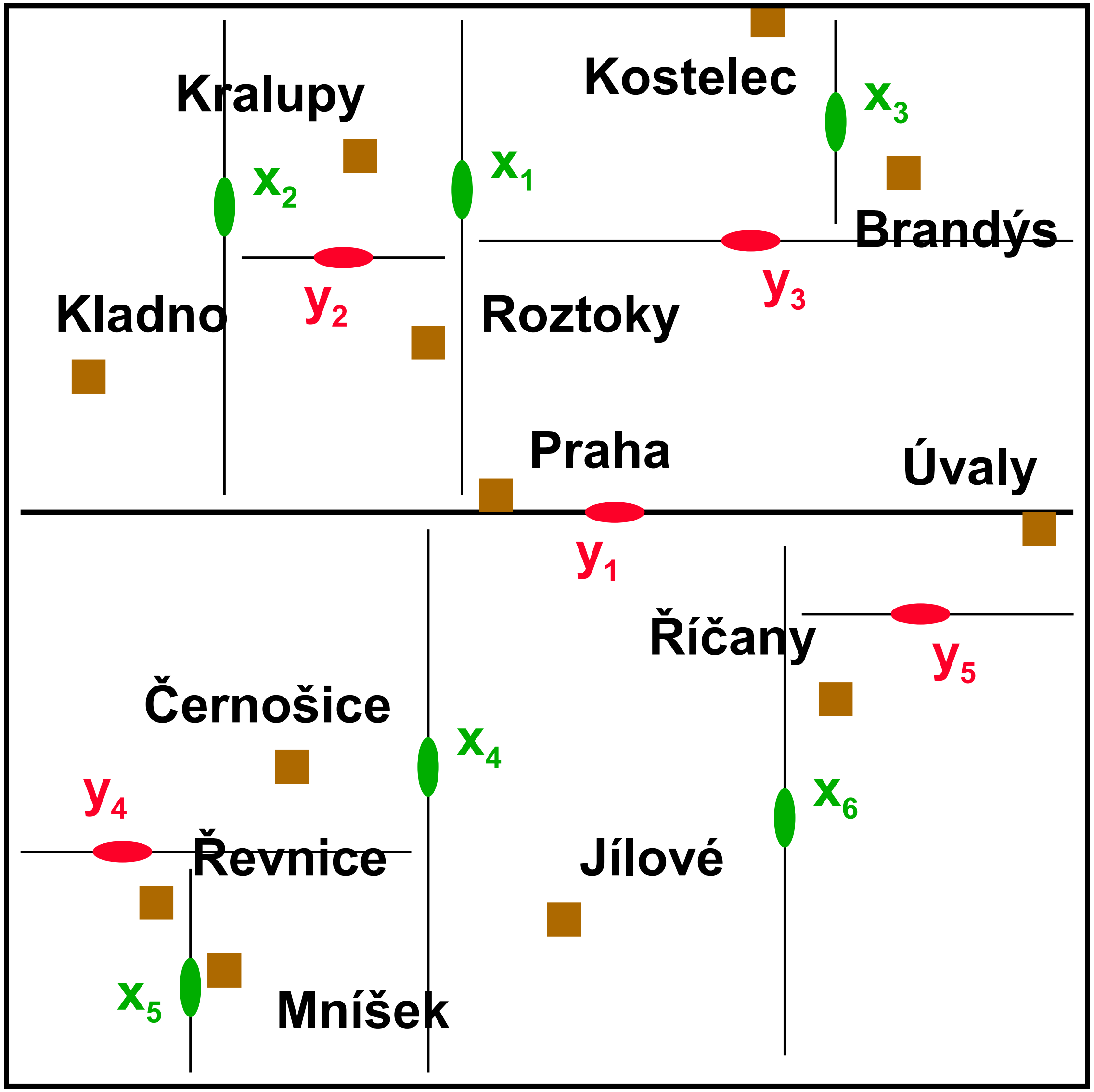


„K-D tree"

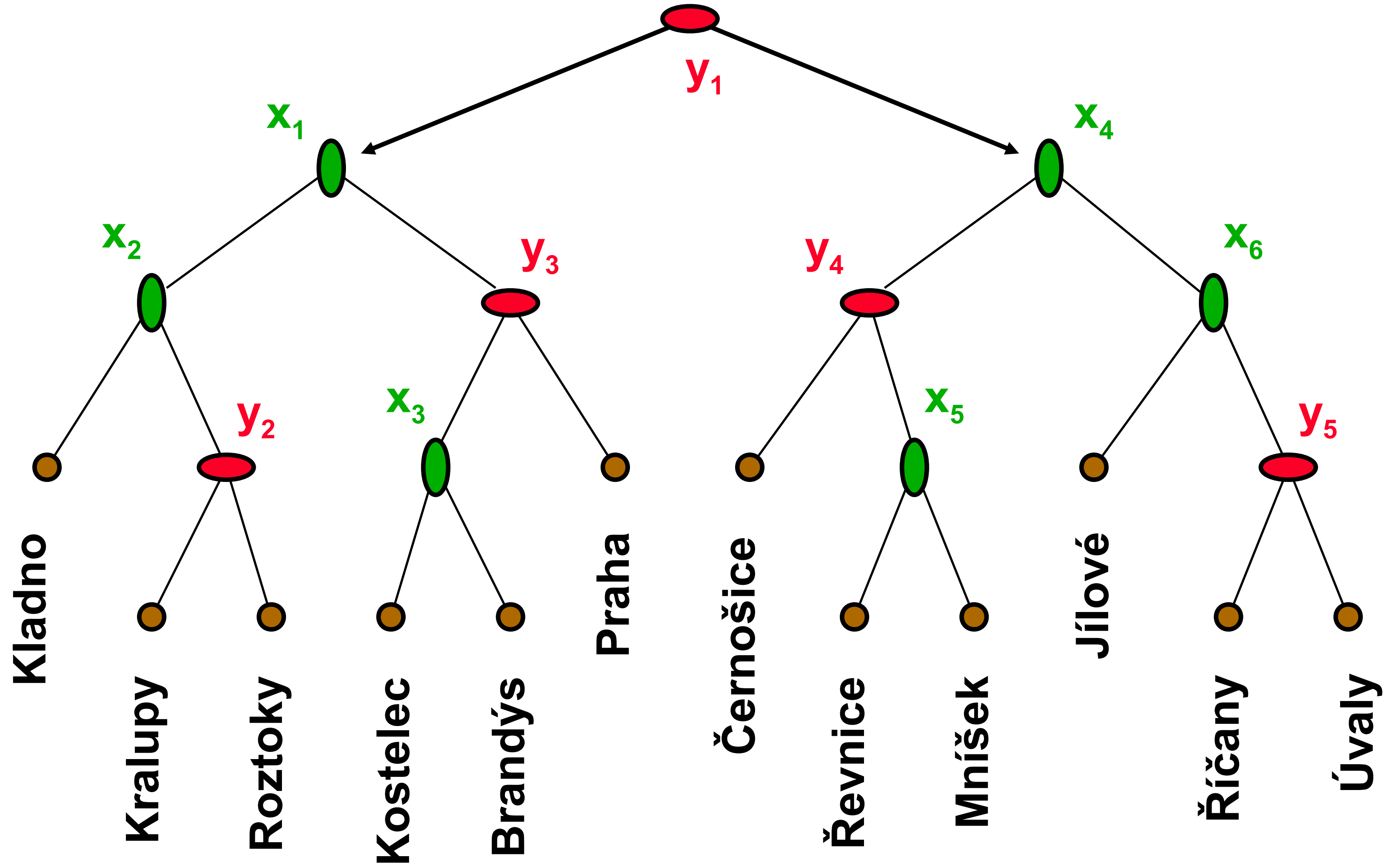

Praha
y
Roztoky
Jílové
Kladno
x
Brandýs
Řevnice
x
Říčany
y
y
y
y
Kralupy
x
Kostelec
x
x
Černošice
x
Mníšek
x
Úvaly

„Adaptive K-D tree" (Friedman)

- representation <u>of point objects</u>
- adaptive splitting - <u>one coordinate at a time (binary tree)</u>. Exact object coordinates not used
- object information stored <u>in leafs only</u>

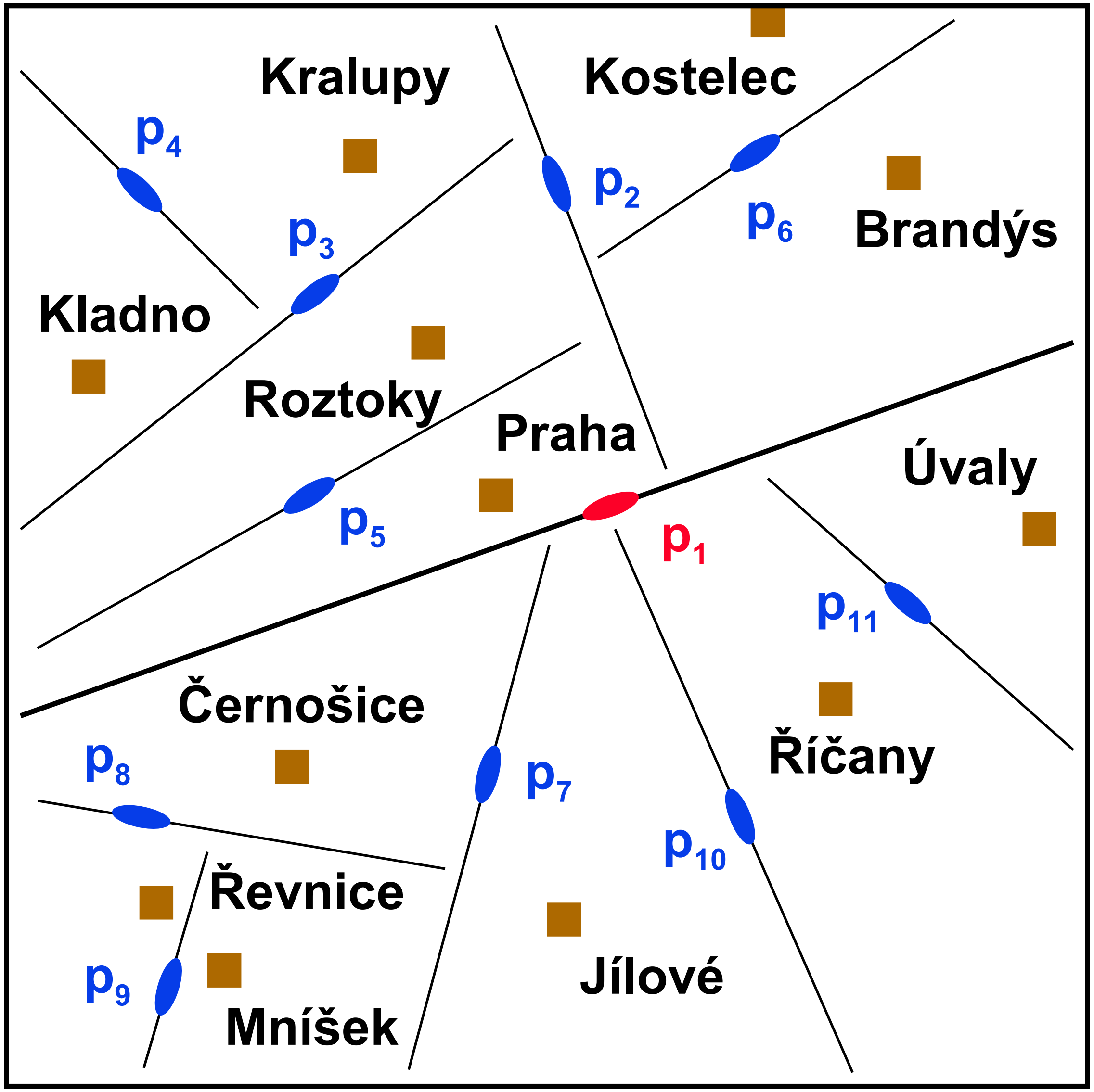


„Adaptive K-D tree"

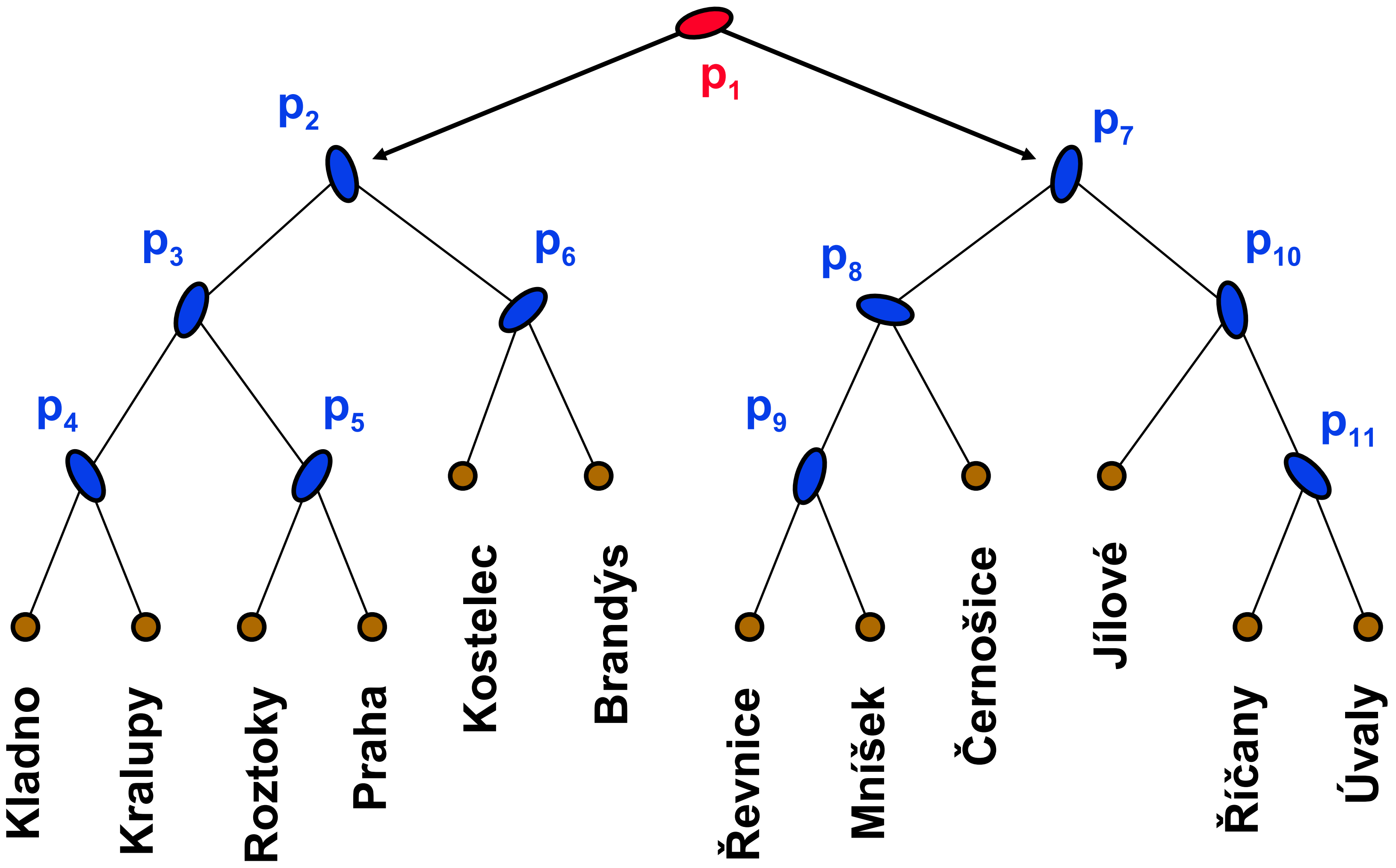


„BSP tree” (Fuchs, Kedem, Naylor)

- decomposition of a plane into convex regions
- adaptive splitting by arbitrary hyperplane
- object information stored in leafs only



„Point BSP tree"

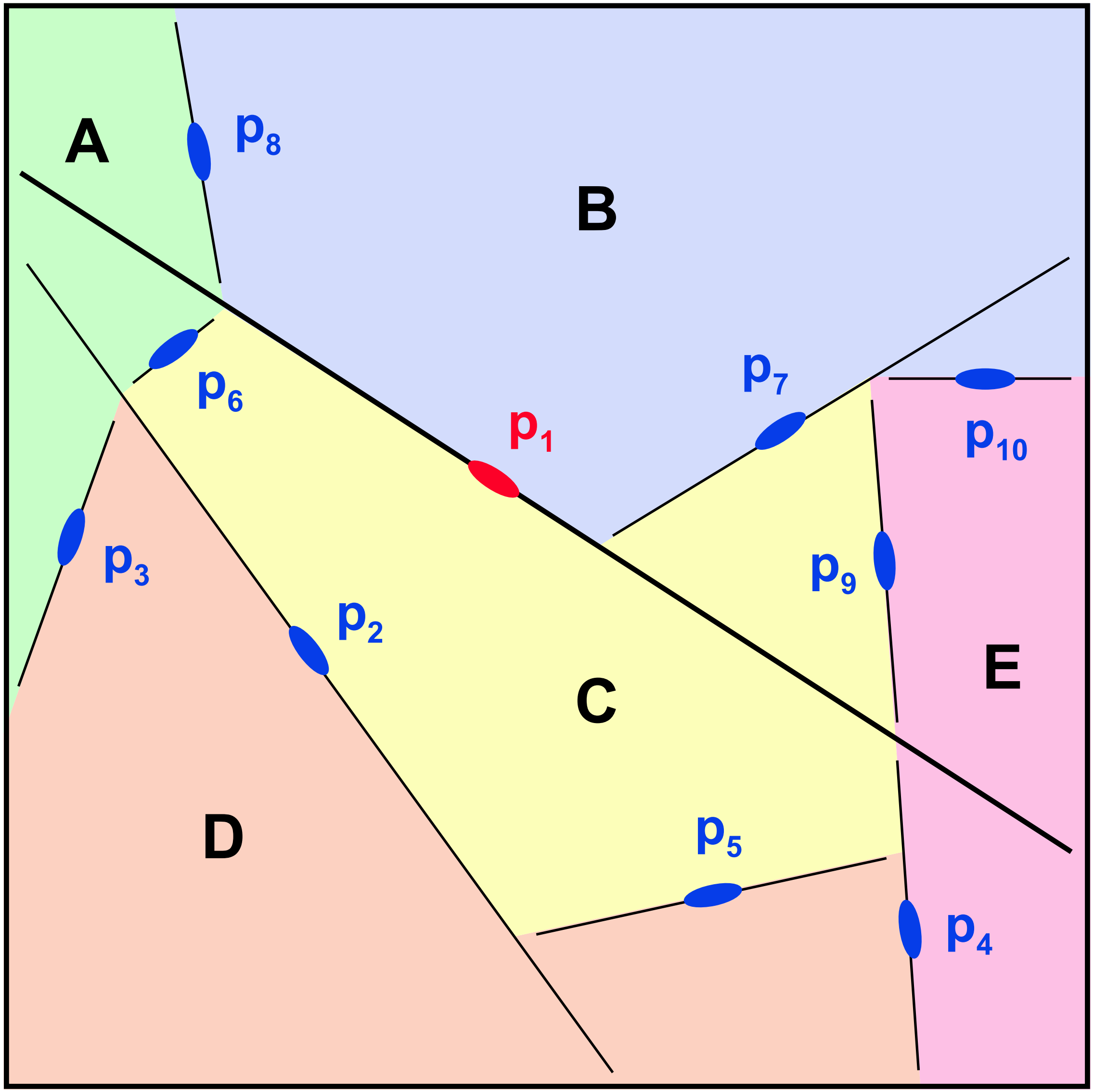


„Region BSP tree”

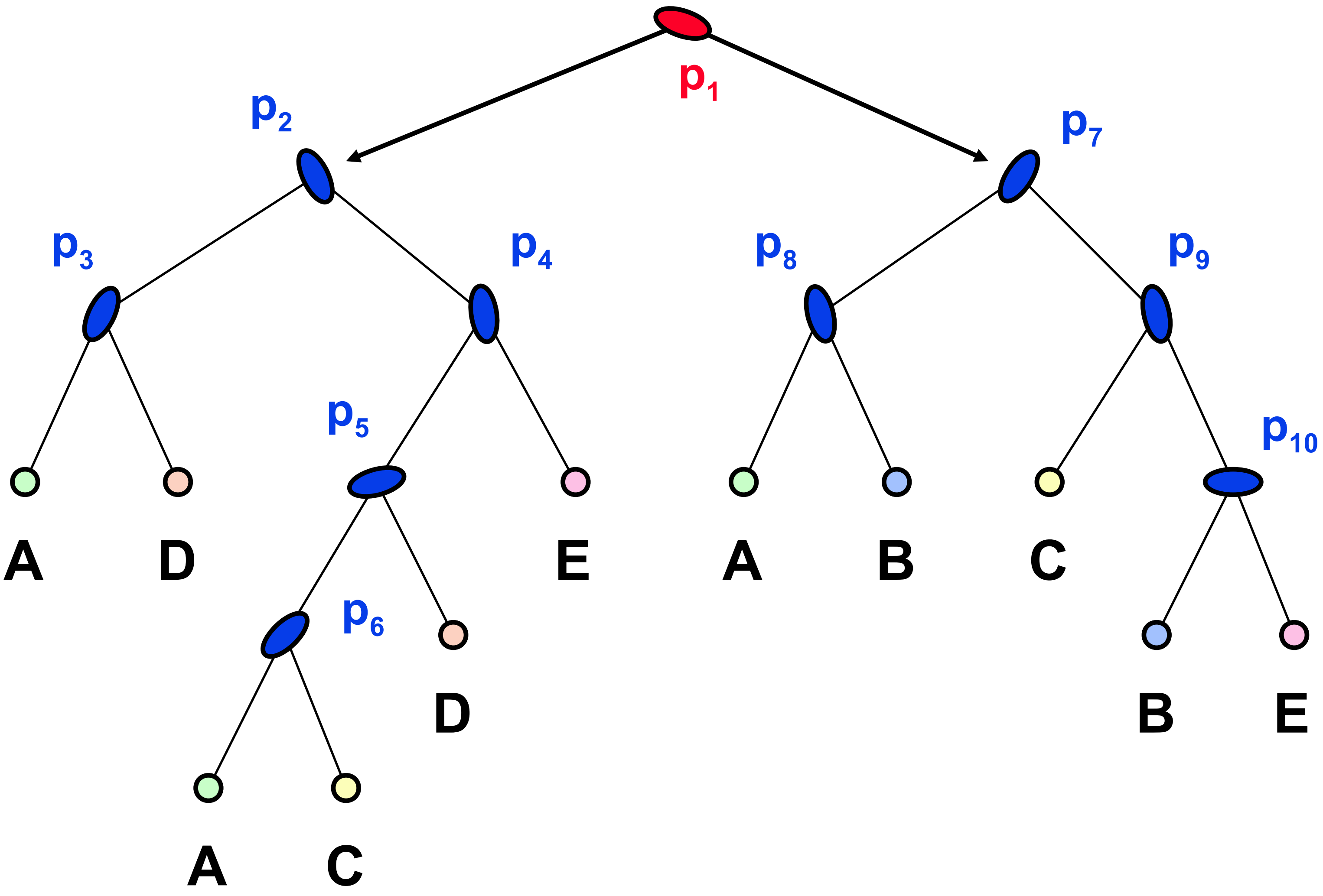
- **polygonal decomposition** of 2/3D space
  - individual regions (polygons/polyhedra) need not be convex
- **polygonal 2/3D scene** representation
  - inner nodes store information about boundary edges/faces
- **set operations** on polygons/polyhedra
  - CSG → B-rep conversion, shadow casting (shadow volumes)

„Region BSP tree”

- **polygonal decomposition**
- **arbitrary splitting hyperplanes (nodes could store information about edges/faces)**
- **polygon identif. in the leafs**



„Region BSP tree”

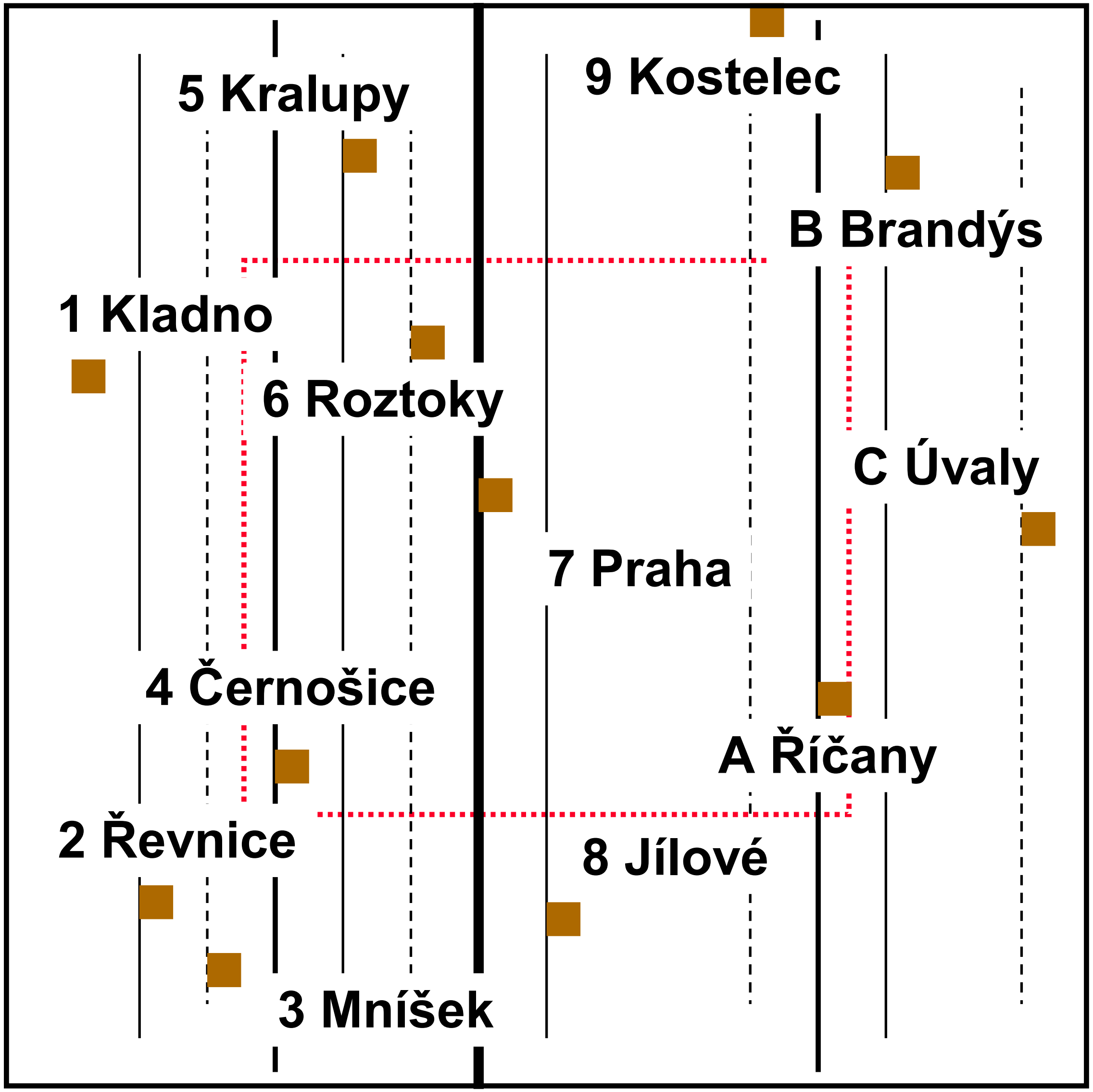


„Range trees”

- efficient implementation of **range queries**
  - $\langle \mathbf{x_{min}}, \mathbf{x_{max}} \rangle \times \langle \mathbf{y_{min}}, \mathbf{y_{max}} \rangle$ ve 2D
  - complexity: $\mathbf{O(\log_2 N + F)}$ in 1D, $\mathbf{O(\log_2^2 N + F)}$ in 2D
- **„1D range tree”**
  - balanced binary search tree, leaves are connected by a double-linked list
- **„2D range tree”**
  - balanced binary search tree for the **x** coordinate
  - every inner node contains „1D range tree” (**y** coord.) for all points in the relevant region

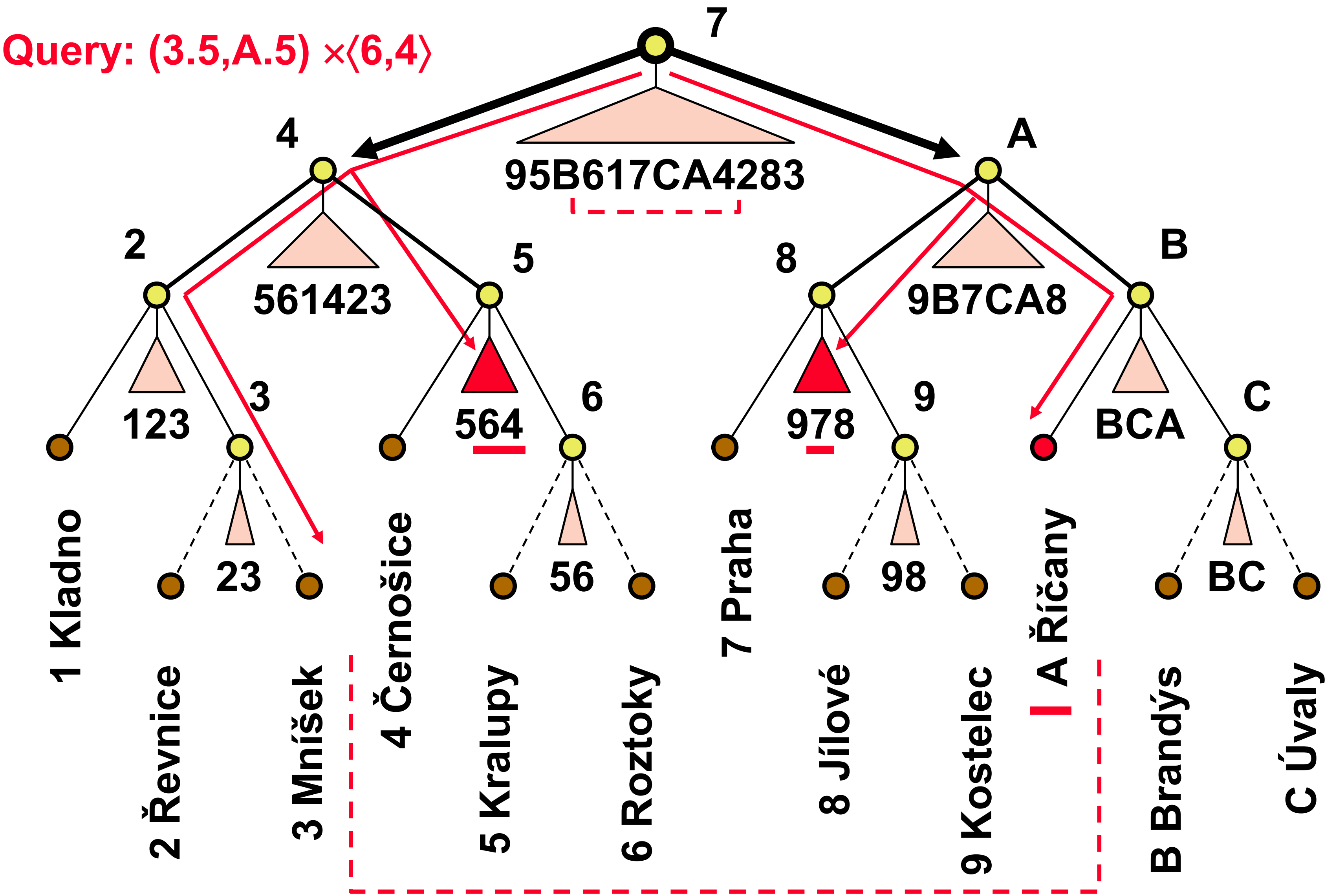
„2D range tree"

- balanced binary tree for x-coordinate
- object information in leaves only
- inner nodes: 1D range trees for y-coordinate



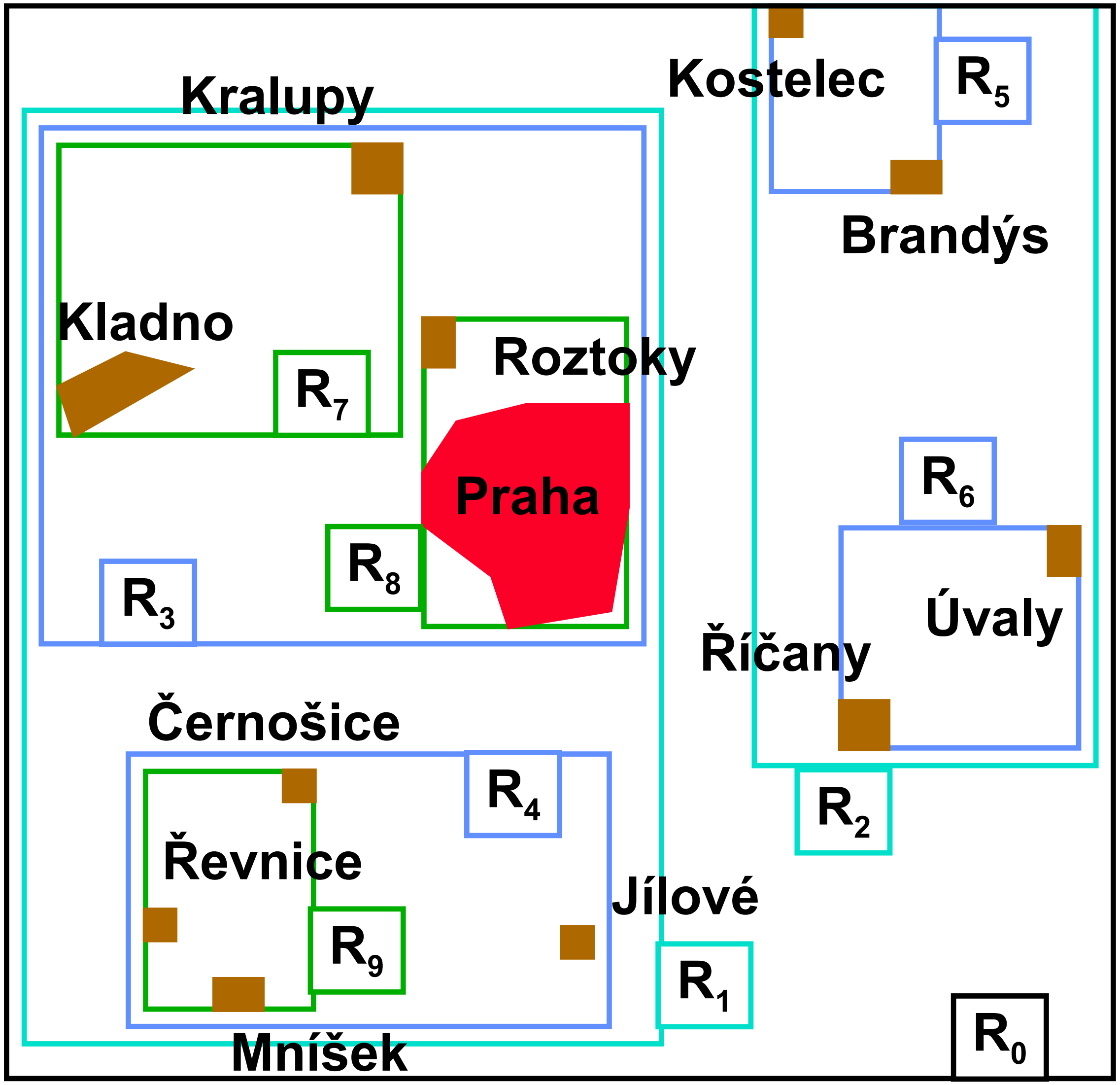
„2D range tree”

Query: (3.5,A.5) ×⟨6,4⟩

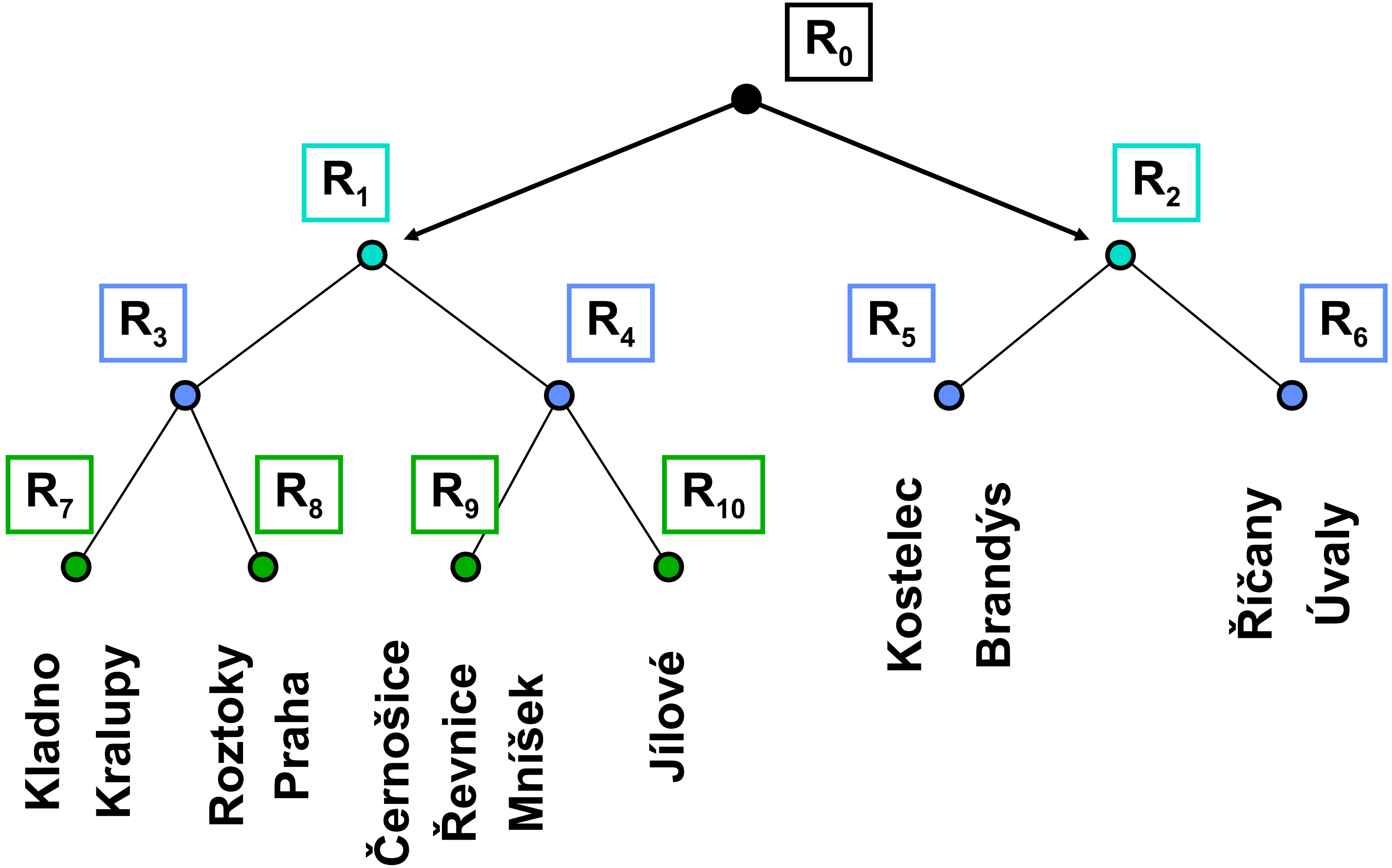


„R-tree” (Guttman)

- objects can be <u>areal</u>
- <u>bounding boxes</u> in inner nodes (the whole subtree must fit into the bounding box)
- object references <u>in leaf nodes</u>



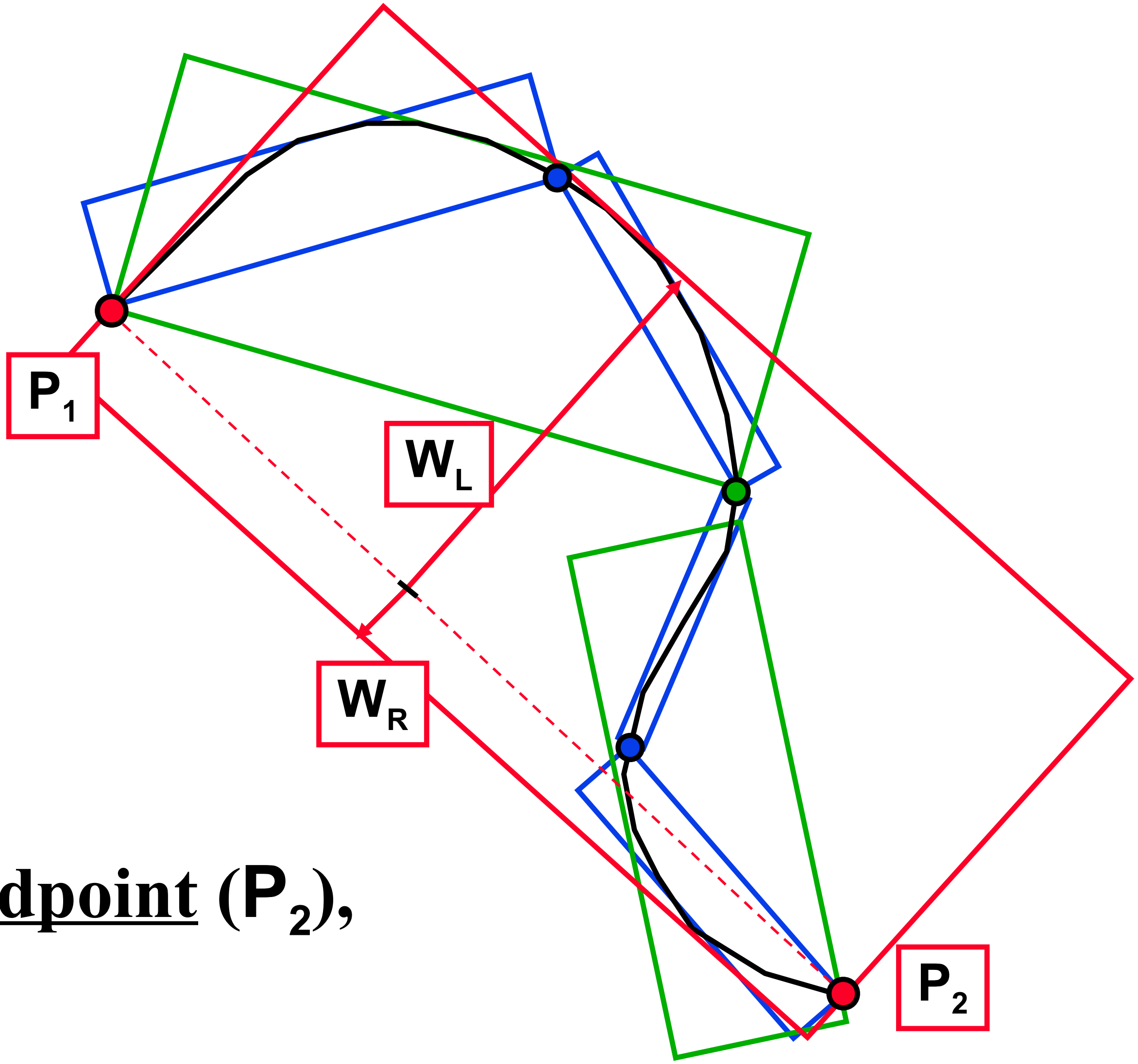
„R-tree”



R_0
R_1
R_2
R_3
R_4
R_5
R_6
R_7
R_8
R_9
R_{10}
Kladno
Kralupy
Roztoky
Praha
Černošice
Řevnice
Mníšek
Jílové
Kostelec
Brandýs
Říčany
Úvaly

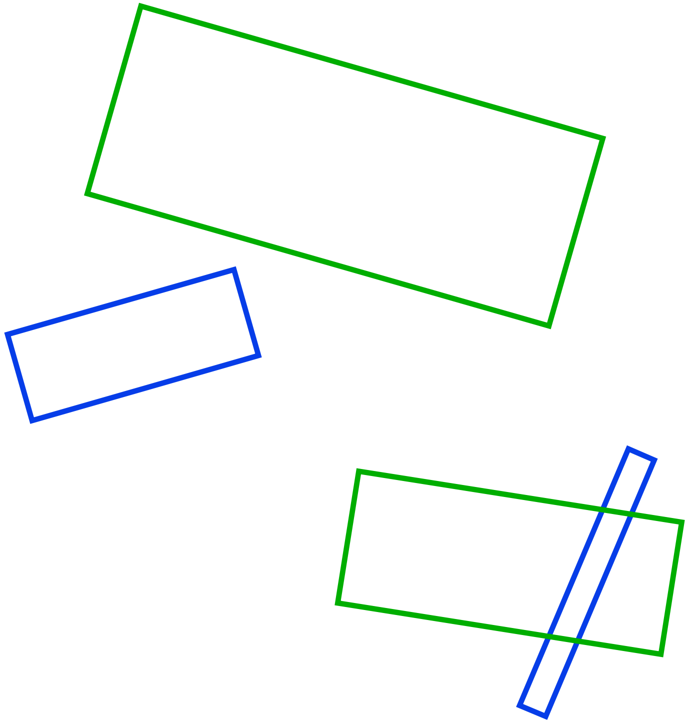
„Strip tree” (Ballard)

- **planar curve** (polyline)
- **adaptive splitting induced by the curvature**
- **oriented bounding rectangle defined by: starting point (P$_1$), endpoint (P$_2$), widths (W$_L$, W$_R$)**

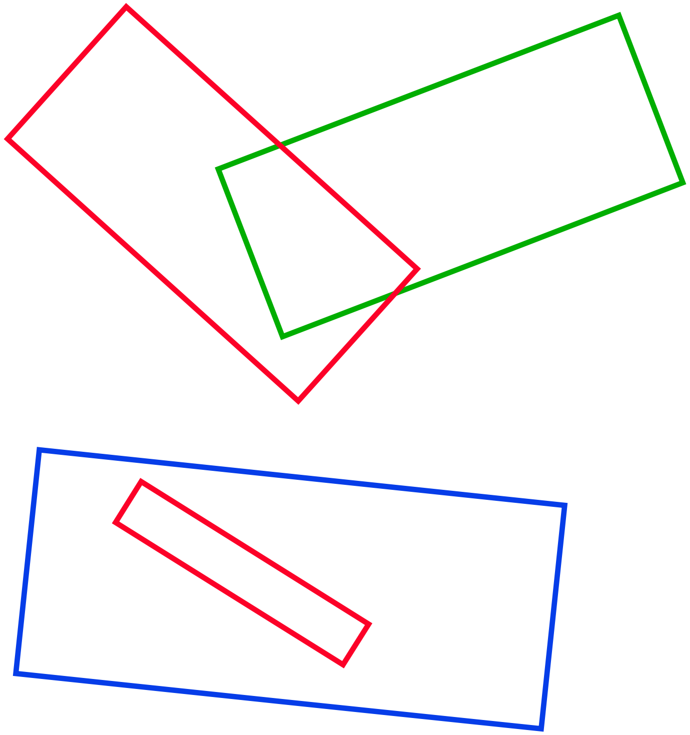


Intersection of two curves

- „**strip trees**” were built for both curves
  - **trivial cases**



  - **subdivision is necessary**

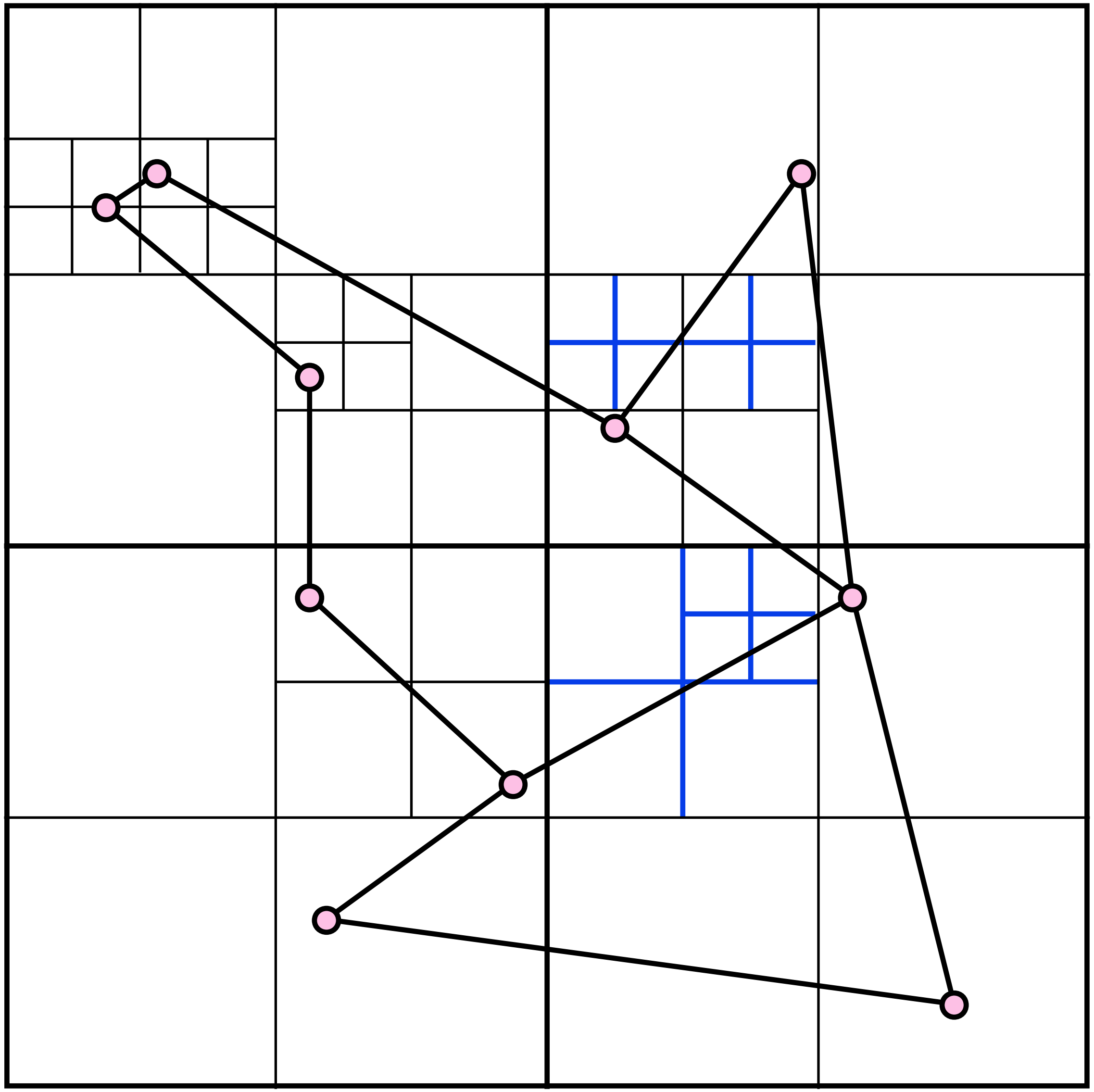


Bounding system hierarchies

- „**Sphere tree**” (Palmer, Grimsdale, 1995)
  – simple tests & transform, approximation not so good
- „**AABB tree**” (Held, Klosowski, Mitchell, 1995)
  – simple test, more complicated transform
- „**OBB tree**” (Gottschalk, Lin, Manocha, 1996)
  – simple transform, more complicated test, good approximation
- „**K-dop tree**” (Klosowski, Held, Mitchell, 1998)
  – complicated transform & test, excellent aproximation

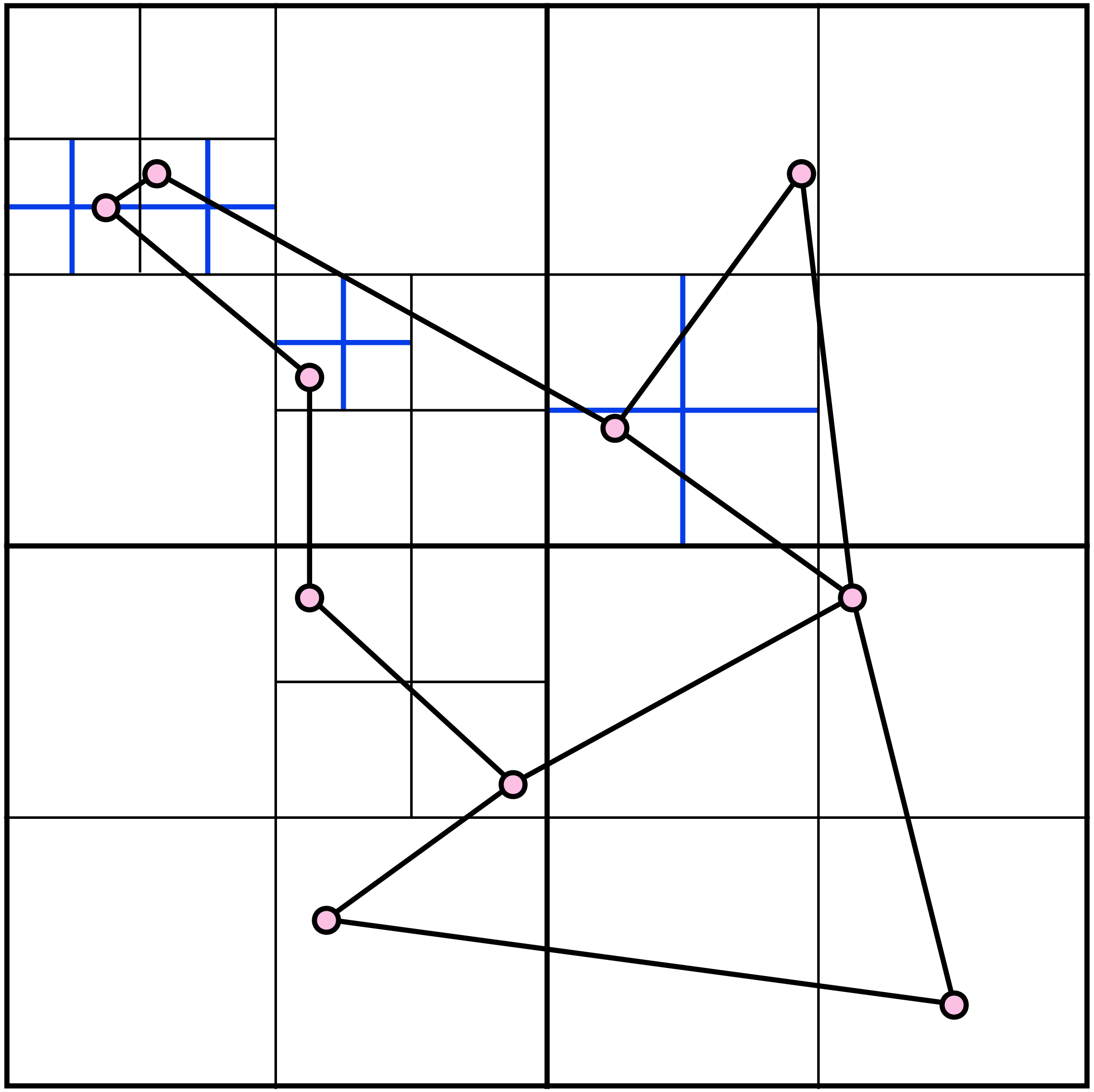
„PM$_1$ quadtree” (Samet, Webber)

- **polygonal maps**
- **max. one vertex per leaf node**
- **more edges in one leaf only if they share common vertex (stored in that leaf node)**



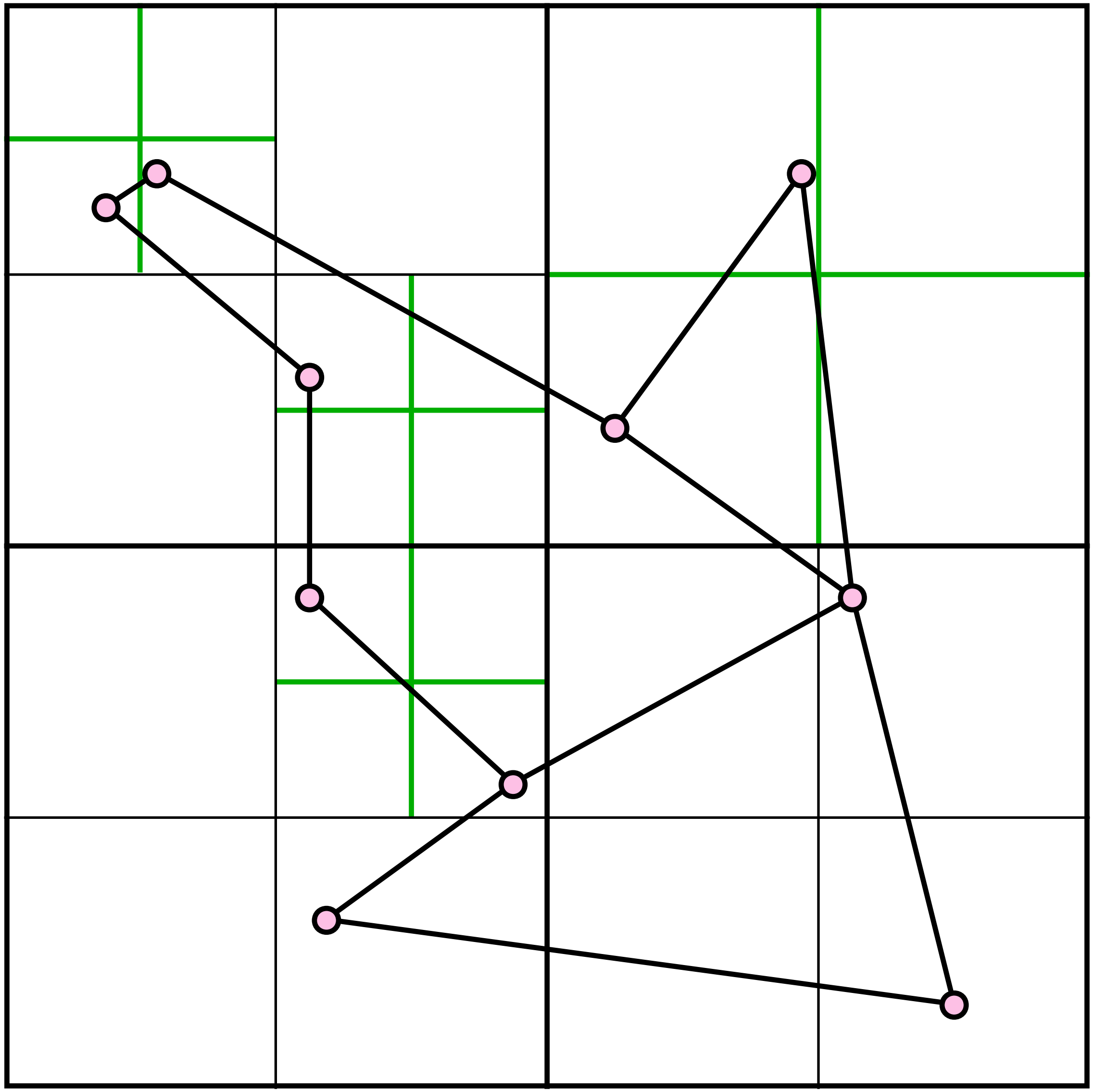
„PM$_2$ quadtree"

- **polygonal maps**
- **max. one vertex per leaf node**
- **more edges in one leaf only if they share common vertex (anywhere)**



„PM_3 quadtree”

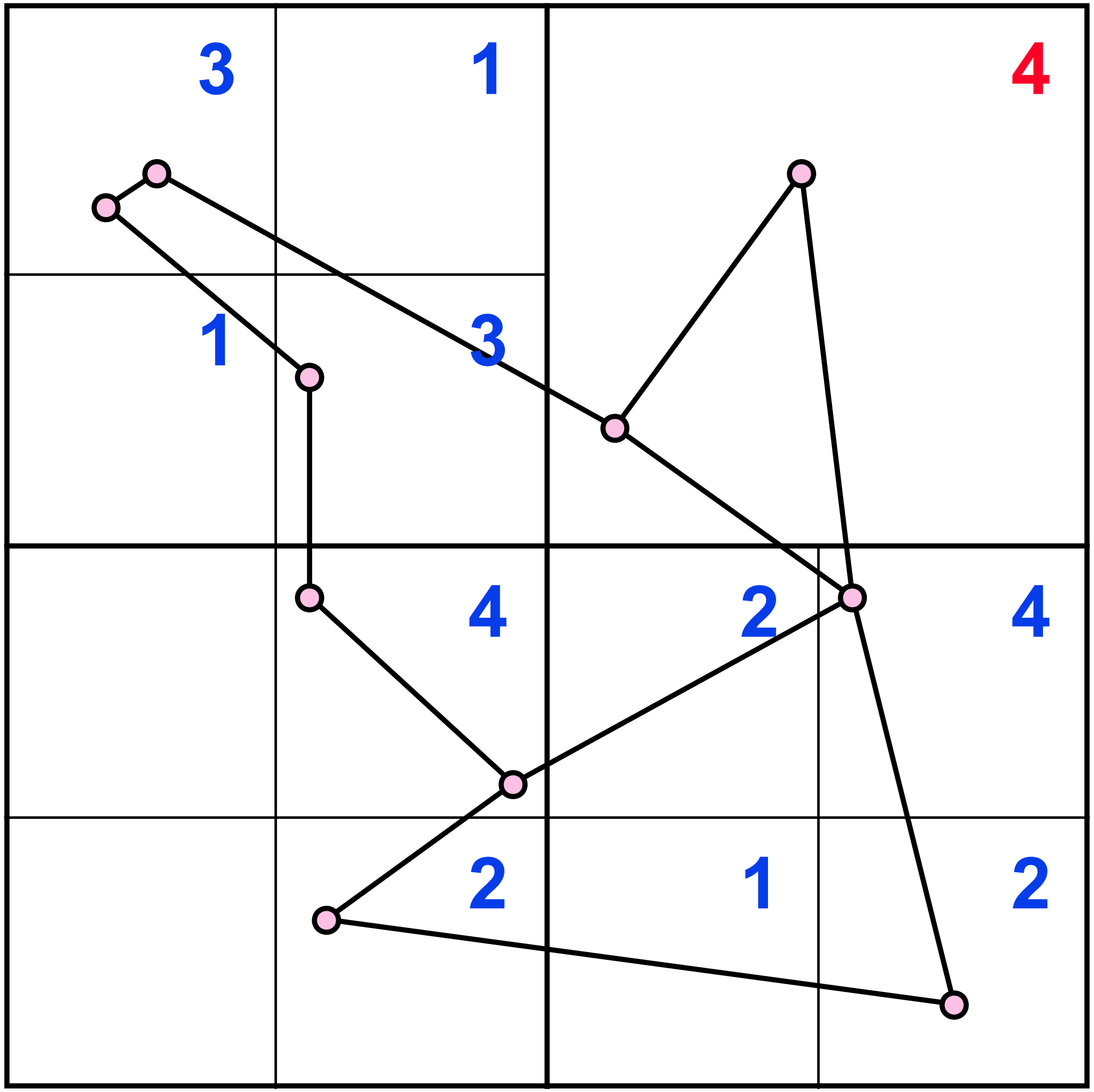
- polygonal maps
- max. one vertex per leaf node
- leaf node can store arbitrary number of edges („PR quadtree” with additional edges)



„PMR quadtree”

- **polygonal maps**
- **max. N edges per leaf node**
- **leaf can store up to 2N references to vertices**

N = 4



Directional pass

- data pass using specific **directional order**
  - visibility (front-to-back or vice versa) in orthographic projection
  - „plane sweep" pass („sweeping")

- pass from the **center point**
  - visibility (form-factors, ..) in perspective projection
  - kNN search („k-Nearest Neighbors")

- **ray-scene intersections** in 3D, etc.

Presumption

- **hierarchical** space decomposition
  - inclusion conditions on objects only, not needed for bounding boxes (e.g. „strip tree”)
- effective computation of **minimal distance of each cell / box** … **d(B)**
  - distance from sweep plane or center point of the pass
  - distance need not be <u>an infimum</u> (but has to be an <u>lower bound</u>)

Algorithm

- auxiliary data structure **H** (heap)
  - efficient operations: **Min**, **DeleteMin**, **Insert**

❶ put the root node into **H**

- **d(B)** is used for **heap sorting**

❷ if **Min(H)** is **an object**, process it and remove it

❸ if **Min(H)** is **a hierarchy cell**, remove it and put all its children back into the heap

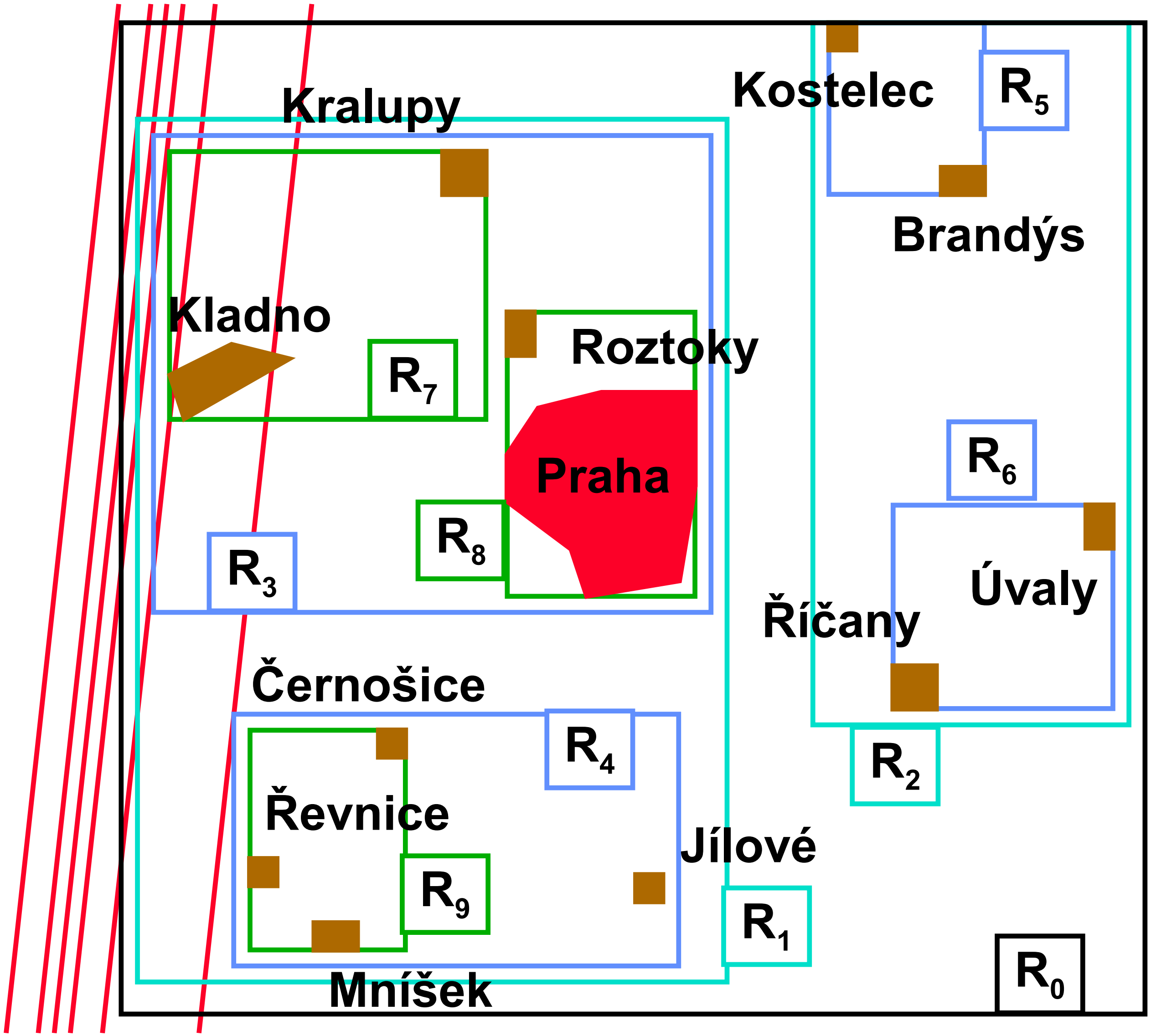
❹ repeat steps ❷ and ❸ until the heap **H** is empty or until all required output objects are processed (kNN)

Example I

- $\{R_0\} \rightarrow \{R_1, R_2\}$
- $\{R_1, R_2\} \rightarrow \{R_3, R_4, R_2\}$
- $\{R_3, R_4, R_2\}$
  $\rightarrow \{R_7, R_4, R_8, R_2\}$
- $\{R_7, R_4, R_8, R_2\}$
  $\rightarrow \{$**Kladno**$, R_4,$
  Kralupy$, R_8, R_2\}$

❶ **Kladno**

- $\{R_4,$ **Kralupy**$, R_8, R_2\}$
  $\rightarrow \{R_9,$ **Kralupy**$, R_8,$
  Jílové$, R_2\}$



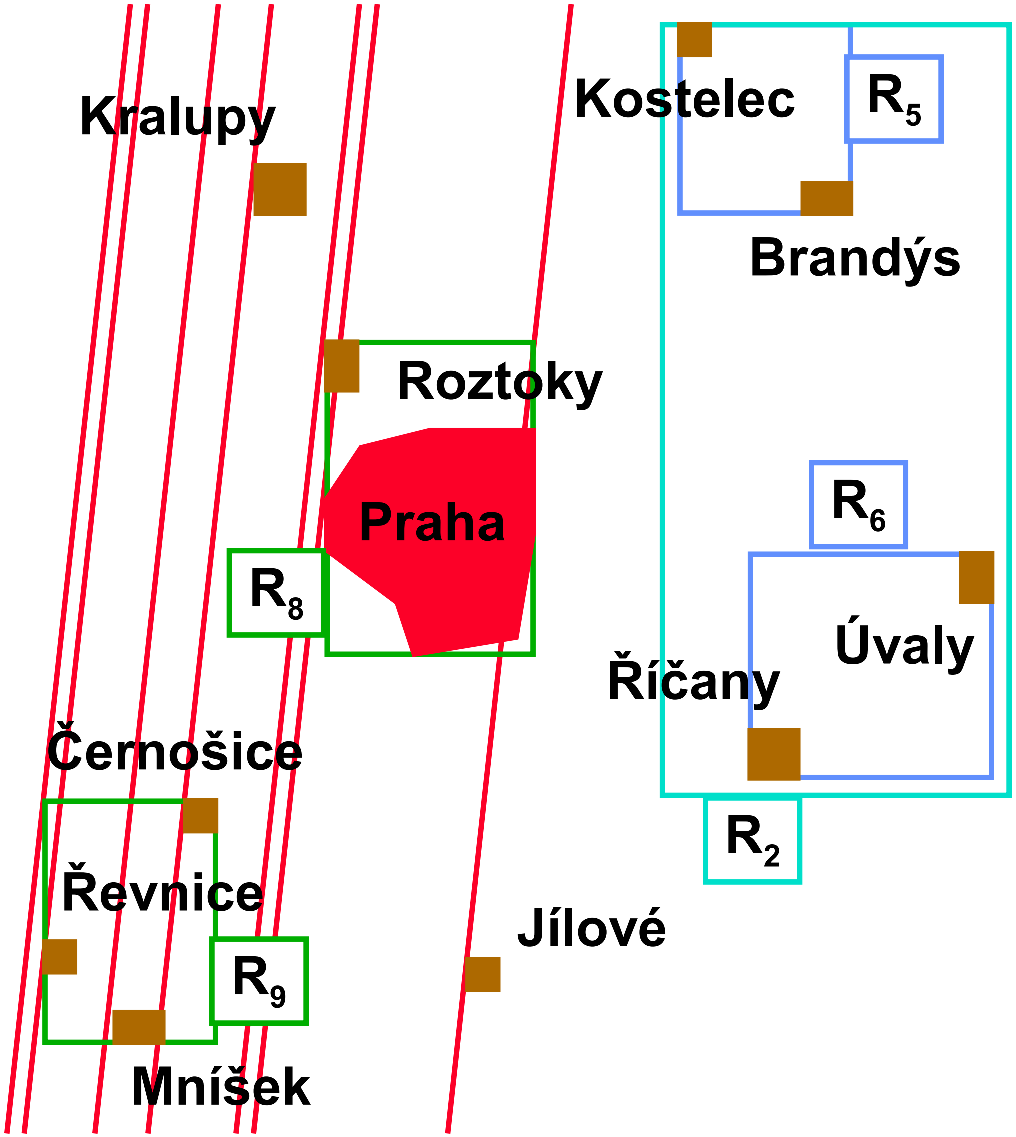
Example II

- {R_9,Kralupy,R_8,Jílové,R_2}
  →{Řevnice,Mníšek,
  Kralupy,Černošice,
  R_8,Jílové,R_2}

❷-❺ Řevnice,Mníšek,
Kralupy,Černošice

- {R_8,Jílové,R_2}
  →{Roztoky,Praha,
  Jílové,R_2}

❻-❽ Roztoky,Praha,
Jílové



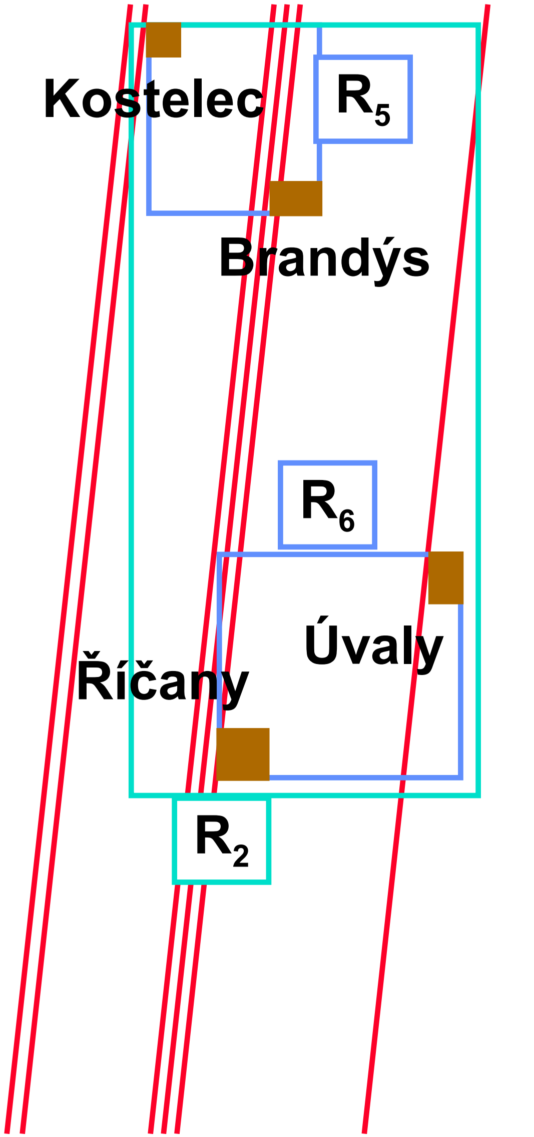
Example III

- $\{R_2\} \rightarrow \{R_5, R_6\}$
- $\{R_5, R_6\} \rightarrow$ {Kostelec, R_6, Brandýs}

❾ Kostelec

- {R_6, Brandýs} → {Brandýs, Říčany, Úvaly}

⑩-❶❷ Brandýs, Říčany, Úvaly



The End

More information:

- **H. Samet**: ***The Design and Analysis of Spatial Data Structures***, Addison-Wesley, 1990
- **H. Samet**: ***Foundations of Multidimensional and Metric Data Structures***, Morgan Kaufmann, 2006
- **F. Preparata, M. Shamos**: ***Computational Geometry, An Introduction***, Springer-Verlag, 1985